

Using Emacs as a Lisp IDE

Bill Clementson
Boulder, Colorado, USA
+1-303-473-1941

bill_clementson@yahoo.com

ABSTRACT

Many Lisp developers use Emacs as their primary tool when developing Lisp code. However, for beginners, coming to grips with how to setup and use Emacs as a Lisp Interactive Development Environment (IDE) can be a formidable task. This paper will walk through a number of the common options for setting up and using Emacs as a Lisp IDE.

Keywords

Emacs, Lisp, IDE, ILISP, ELI, Inferior Lisp Mode, Common Lisp.

1. WHY USE EMACS

There are a number of different good editors available with some of the commercial Common Lisp products and a wide variety of commercial and open source editors that can be used for writing CL code. So, why do many CL developers choose to use Emacs instead of one of the alternatives? Some of the more common reasons are:

- Emacs has fantastic support for working with Lisp code. It supports automatic code indenting, code coloring, s-expression editing operations, and much more.
- By using Emacs, you aren't tying yourself into the editor supplied by a single CL vendor.
- Emacs runs on virtually every OS and can be made to work with virtually every CL implementation. Since it works consistently across different operating systems, many people find that it provides them with a consistent working environment regardless of the OS that they might need to use.
- Emacs is extensible. You can configure Emacs and program your own extensions in Emacs Lisp, a Lisp dialect that is very similar to CL.
- Built-in support for different source code version control systems.

- Emacs is much more than just an editor and can be customized to do many common tasks (e.g. – reading mail and news, shell tasks, documentation, internet browsing) as well as interact with other tools that you may choose to use for some tasks.
- There are a vast number of add-on packages that extend its functionality and that are freely available.
- Emacs will probably always be around so it is the last editor you'll ever "have" to learn.
- Emacs works well either with a mouse or without a mouse. For editing code, it is quite important to be able to keep your hands on the keyboard – any time you have to take your hands from the keyboard to do some action, you are slowing down your code entry and disrupting the natural flow. However, sometimes when you are just browsing code, using a mouse is more productive. Emacs suits both styles.
- Emacs has a large user base with multiple newsgroups providing a very high level of support for both beginners and experienced users.
- Like most sophisticated tools, Emacs requires an effort to learn well. However, once learned, the benefits of using Emacs far outweigh the effort spent in learning it.

2. INTRODUCTION TO EMACS

In order to talk about using the Emacs editor as a Lisp IDE, we need to first establish the level of knowledge that readers are expected to have. We will assume that people reading this paper will have already installed Emacs on their computer and will have received some tutorial instruction in the concepts and basic usage of Emacs. If you fall into this category, you can safely skip to the next section. If not, the following subsections will explain how you can get Emacs and learn some basic usage concepts. In subsequent sections, we will concentrate on those aspects of Emacs that a Lisp programmer will find most useful.

2.1 Obtaining Emacs

Emacs can be downloaded from the GNU site [1] or from any of the many mirrors on the Internet. Alternatively, a CD [2] can be purchased from the Free Software Foundation with Emacs on it.

2.2 Learning Emacs

Learning Emacs is not a task that we will spend much space on in this paper. In fact, since Emacs is an extensible editor, one could argue that it is not possible to ever learn all of Emacs. A basic Emacs tutorial is included with the default installation and can be accessed by pressing "C-h t" (press and hold down the Control key, press the "h" key, release both, press the "t" key). The Emacs

tutorial should be the first tutorial for anyone trying to learn Emacs. For those who learn better with a book, there are a number of options available [3]. For the impatient, Bob Rogers has some basic introductory Emacs material on the Web [4] and it is possible to get a quick start on key Emacs commands by reading through his tutorial material (there are other online Emacs tutorials – Google can be helpful here). In addition, the Emacs Wiki can be a valuable resource of information on all things having to do with Emacs [5].

Learning Emacs Lisp (elisp) can be very useful as well, in order to customize your Emacs environment and change configurations. Elisp is very similar to CL. The primary differences in Elisp are:

- Dynamic scope is everywhere – there is no lexical scope for variables
- No package system is supported
- There are no reader (or reader-related) functions
- Emacs Lisp doesn't support all the types that are supported in CL
- Incomplete implementation of CLOS (the add-on EIEIO [6] package provides some functionality but it is an incomplete CLOS implementation)
- Not all of CL is supported (although the standard cl feature that comes with Emacs adds a lot of CL support)
- No numerical tower support

However, many people have come to CL via Emacs and Emacs Lisp, so the differences are far smaller than the similarities. A good introduction to programming in Emacs Lisp is [An Introduction to Programming in Emacs Lisp](#) [7] which is available in a number of different formats. In addition, the O'Reilly book [Writing GNU Emacs Extensions](#) [8] is a good introduction to Emacs Lisp programming.

2.3 Getting Help in Emacs

Emacs help is quite extensive. There are many different help commands bound to the “C-h” prefix. The “C-h ?” binding will give a list of the help keys that are available. Some of my most commonly used help keys are:

“C-h a” is the key binding for “command-apropos”. Type a substring, and see a list of commands that contain that substring.

“C-h b” is the key binding for “describe-bindings”. It displays a table of all key bindings.

“C-h f” is the key binding for “describe-function”. Type an elisp function name and get documentation of it.

“C-h i” is the key binding for “info”, the info documentation reader. This is useful for reading both standard Emacs documentation and CL-specific manuals.

“C-h k” is the key binding for “describe-key”. Type a command key sequence; it displays the full documentation.

“C-h m” is the key binding for “describe-mode”. Get documentation of the current minor modes, and the current major mode, including their special commands.

“C-h w” is the key binding for “where-is”. Type a command name; it prints which keystrokes invoke that command.

In addition, the apropos command (not to be confused with the “C-h a” apropos which only searches for functions that match a substring) is useful for finding any command or variable given a substring.

2.4 Getting used to Emacs

Emacs makes extensive use of modifier keys such as Control, Escape and Alt. On some keyboards, extensive use of these keys can be tiresome and contribute to typing-related injuries such as RSI (Repetitive Strain Injury). In order to make it easier to use Emacs, some users resort to one or more of the following:

- Replacement keyboard: When Emacs was originally designed, many keyboards [9] had the Control key placed in a more convenient position from where it is placed on most PC-style keyboards today. Some people prefer to buy a keyboard that has a key layout with the Control key placed in a more easily accessible position. [10]
- Swap the position of the Control key: By configuring the Caps Lock key as the Control key, you can overcome the awkward placement of the Control key on most modern keyboards. You can then conveniently use your left thumb for Meta/Alt and your left pinky for Control. There are different ways to do this on UNIX®, Mac OS X, and Windows [11].
- Foot pedals: There are a number of different foot pedals [12] that Emacs users sometimes use. These allow modifier keys (or other keys) to be assigned to a foot pedal press rather than a finger press, thus reducing the number of key presses required.
- Redefining keys in Emacs: It is easy to redefine key sequences in Emacs to minimize the number of modifier keys used for frequent commands.

Any (or all) of the above will make it a lot easier to use Emacs; however, whether these options are possible or not depends a lot on the nature of the work environment (or environments) of each individual. In some environments, using alternative hardware may not be an option. Also, redefining keys may not be a good option for a new Emacs user as they will not necessarily be able to make appropriate key choices until they have used Emacs for a while. In any case, many users find the Emacs defaults to be quite usable.

3. LISP MODES IN EMACS

Most people who program in Common Lisp using Emacs use one of three Emacs major modes (a “major mode” in Emacs provides a custom set of features for editing text of a particular sort): Inferior Lisp Mode, ILISP, or ELI. The following subsections will give a brief summary of the features, pros and cons of each of these.

3.1 Inferior Lisp Mode

Inferior Lisp Mode is based off of comint mode, a major mode designed for interacting with an inferior (inferior in the sense that it is controlled by Emacs) interpreter (e.g. – Shell, Scheme, Common Lisp, etc.). Although comint mode provides some base functionality, the Lisp-specific functionality is provided by the additions in Inferior Lisp Mode.

3.1.1 Inferior Lisp Mode - Pros

Comes with Emacs, fast start-up, easy setup. Supports many Lisp implementations.

3.1.2 Inferior Lisp Mode - Cons

Limited functionality (compared to ILISP and ELI). No multiprocessing support (e.g. – the ability to have multiple Lisp listeners all using the same Lisp image).

No multiprocessing support.

Since Inferior Lisp Mode is based on comint mode, there are sometimes conflicts with comint mode key bindings or functionality.

3.1.3 Inferior Lisp Mode - Setup

Included with Emacs, so no separate installation required. No configuration is necessary, simply run the Emacs command '(run-lisp "lisp-program")', substituting the executable name of a Lisp executable for "lisp-program".

3.2 ILISP

ILISP was written as a replacement for the standard Inferior Lisp Mode. It is also based on comint mode but includes influences from a number of different Lisp environments (including Symbolics, Thinking Machines and CMU Common Lisp). It has the following main features (taken from the ILISP manual) over and above what is offered in Inferior Lisp Mode:

- Evaluation and compilation of an entire file, or of a region, a definition, or an s-expression of a buffer. The user can specify for ILISP to switch to the inferior Lisp buffer after evaluation/compilation or to stay in the source buffer.
- Evaluation and compilation can be done either synchronously (ILISP waits for the answer), asynchronously (ILISP does not wait), or in batch mode.
- Arglist, documentation, describe, inspect and macroexpand.
- Access to Common Lisp HyperSpec [13] and CLtL2 [14].
- Find source both with and without help from the inferior Lisp, including CLOS methods, multiple definitions and multiple files.
- Edit the callers of a function with and without help from the inferior Lisp.
- Dynamically sized pop-up output windows that can be scrolled or removed from any window.
- When the user sends an expression from a Lisp source buffer for evaluation in an inferior Lisp process, ILISP automatically switches to the package that is indicated at the beginning of the buffer. The expression is therefore read by the inferior Lisp process with the correct current package.
- Trace/untrace a function.
- "M-q" ("Fill-paragraph") works properly on paragraphs in comments, strings and code.
- Find unbalanced parentheses.
- ILISP has commands for closing all open parentheses.
- Uniform interface to Lisp debuggers.
- All ILISP commands are accessible from a menu.

3.2.1 ILISP - Pros

Vastly superior to Inferior Lisp Mode in functionality. Supports many Lisp implementations.

ILISP is managed as an open-source project on SourceForge and there are a group of developers who actively support it.

3.2.2 ILISP - Cons

No multiprocessing support.

Since ILISP is based on comint mode, there are sometimes conflicts with comint mode key bindings or functionality.

3.2.3 ILISP - Setup

A basic installation involves downloading the ILISP package from the web [15], building it and configuring it. The ILISP documentation (and sample .emacs file) helps to guide the user through this process; however, ILISP installation/setup is a common source of problems for new ILISP users. In addition, there are a lot of configuration options which, although providing a very nice level of standard customization, can confuse new users.

There are a number of useful web sites [16] that provide step-by-step instructions for ILISP installation. New users are better off following the setup/configuration instructions on these sites and subsequently tailoring ILISP as they become more used to using it.

On a Debian Linux system, installing ILISP is a snap – one command is all that is needed.

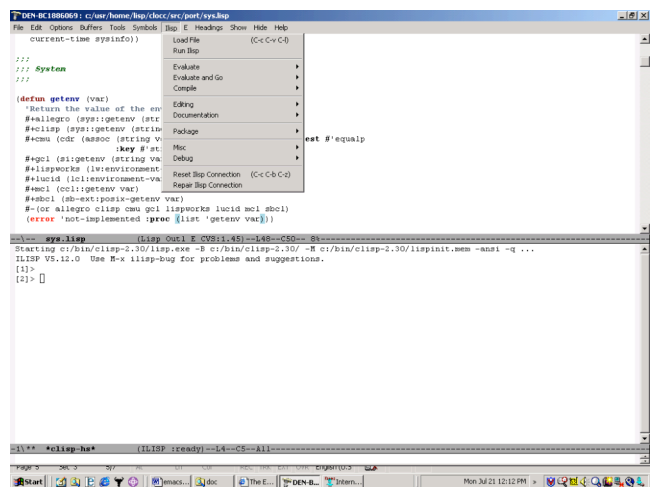


Figure 1. ILISP menu with listener window

3.3 ELI

ELI stands for "Emacs-Lisp Interface" and was developed by Franz as a means of using Emacs with Franz's Allegro Common Lisp (ACL) product. In contrast to Inferior Lisp Mode and ILISP (both of which were built on top of the existing comint mode), Franz created their own socket-based communication protocol (LEP or "Lisp Editor Protocol") for communicating between Common Lisp and Emacs, using a combination of multiprocessing threads on the CL side and process filters in Emacs (Emacs does not support multiprocessing).

With the multiprocessing support in ELI, it is possible to have multiple Lisp listeners (all using the same Lisp image) as well as background streams. In addition to being a significant productivity enhancer, this functionality allows for easier

debugging of certain types of errors (e.g. – in ILISP if you get an error in a thread that is not running in the listener, it is hard to deal with the error. In ELI, this is not a problem). Also, you can have Lisp code running in a listener, make a change to a function in an Emacs buffer, evaluate it, and that function will now be available in the running code in the listener.

Although ELI was developed by Franz, they released it under a GPL license, thus allowing others to port it to other CL implementations. Some limited porting has been done to allow ELI to be used on Sciener CL [17], CMUCL [18], and SBCL [19].

3.3.1 ELI - Pros

ELI is roughly equivalent to ILISP in functionality but does not depend on comint mode (users of ILISP and Inferior Lisp Mode sometimes have conflicts with either comint mode functionality or key bindings).

It supports Franz’s ACL extremely well (no surprise given that ELI was developed by Franz). In fact, Emacs can be configured as the default editor in ACL and it works very well with the other ACL tools.

ELI is the only Emacs Lisp development package that supports multiprocessing (this is a **big pro**).

ELI also has commands that allow you to work with changed definitions. These are useful for keeping track of and working with (e.g. – copying, compiling, evaluating) changes that have been made to Lisp code and is similar to functionality that was available on Lisp Machines.

ELI provides a standard, consistent set of options for managing output (whether the output is from evals/compiles or from ELI requests to Lisp).

Support for ELI is provided by Franz and is very good.

3.3.2 ELI - Cons

For CMUCL and SBCL, only limited functionality is available (not all of ELI has been ported to these platforms) and support is not really available.

There is no built-in support for accessing either Franz or CL documentation directly from ELI; however, a number of add-on packages can be used to accomplish this (see the documentation section later in this paper).

3.3.3 ELI - Setup

Basic ELI setup is very straight-forward for ACL. Since ELI is included with ACL, no separate download is required. A sample .emacs file is included in the ELI examples folder that illustrates the changes that need to be made when activating ELI in Emacs. Documentation is included in the standard documentation that comes with ACL.

For non-Franz CL’s setup is a bit more complicated and full ELI functionality may not be available.

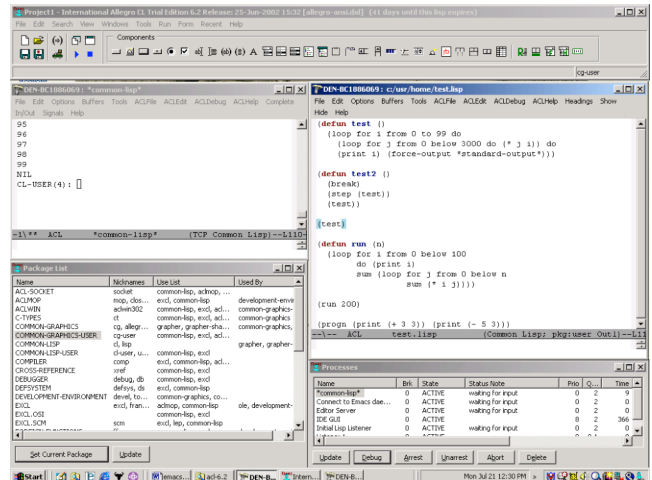


Figure 2. Emacs/ELI with ACL tools

3.4 Which one to use?

Choosing between Inferior Lisp Mode, ILISP, and ELI is not always an easy choice. For the occasional CL programmer, Inferior Lisp Mode is easiest to setup, is always available on any Emacs, and has enough functionality to be productive. However, it has very few frills or productivity-enhancers.

A programmer who intends to use CL more extensively would probably choose either ILISP or ELI because one can be much more productive using either of those options. For ACL, ELI is the best option. For non-Franz Lisp implementations that support multiprocessing, the “best” choice would probably be either a comprehensive port of ELI or a version of ILISP that supported ELI’s LEP protocol (or a similar approach) for multiprocessing. This would provide the best combination of functionality; however, it is not an option that is currently available.

The bottom line: for non-Franz CL’s ILISP is going to be the best choice (at least until/if the ELI ports improve). For Franz’s ACL, ELI is the best option. Inferior Lisp Mode should be considered a “last resort” option.

4. WORKING WITH LISP CODE

Emacs is designed as an extensible editor and (although it can be used for many things) working with text/code is what it does best. Many of the standard text editing features (e.g. – cut, copy, paste, etc.) of Emacs will apply unchanged when working with Lisp code. However, there are a number of special features in Emacs that make working with Lisp code more convenient. This section will discuss these Lisp-specific features.

4.1 SEXP Based Editing

Lisp is made up of s-expressions (sexp’s) and it is nice to be able to work with units of sexp’s when you are editing. Emacs does this very nicely and in a way that is mnemonically easy to remember. Many of the standard command key bindings which apply to characters apply to sexp’s if Meta (Alt) is added to the command. An easy way to think of this is to remember that the Control-Meta combination typically means “apply the command to logically larger units”. For example:

- “C-f” is the command for forward-char, while “C-M-f” is the command for forward-sexp

- “C-k” is the command for kill-line, while “C-M-k” is the command for kill-sexp
- “M-q” is the command for fill-paragraph (which justifies text), while “C-M-q” is the command for indent-sexp (which justifies the sexp according to Lisp indentation rules)

To get a complete listing of sexp-related commands, just press “C-h b” when you are in a lisp source file and search the resulting key binding help buffer for “sexp” (you can use the “occur” command to do this).

All of the commands in this subsection are available regardless of whether Inferior Lisp Mode, ILISP, or ELI is used.

4.2 Miscellaneous Lisp Editing

There are a number of other commands/features that are particularly useful when editing Lisp source code:

- Emacs supports Lisp parentheses very nicely. It will enter both parentheses and position the cursor after the start paren with the command insert-parentheses (“M-("). Emacs automatically (so long as you have “(show-paren-mode 1)” in your .emacs file) does balanced parenthesis flashing, showing you the matching begin paren for each end paren you type. If you are typing a mixture of delimiters (e.g. – (, [, {) and the closing delimiter doesn’t match up with the corresponding opening delimiter, the offending delimiter will be highlighted in a distinctive color. The Emacs command check-parens checks the current buffer to see whether all parentheses are matched up.
- Emacs automatically indents a line of lisp code when you press ENTER. If Lisp code is entered in a different manner (e.g. – pasted from some other location), pressing TAB at the beginning of the line will indent the line to the correct position.
- Emacs Lisp has traditionally had different indentation rules from CL. In order to use CL indentation rules when editing Lisp code, put the following line in your .emacs file:

For Inferior Lisp Mode or ILISP:

```
(add-hook 'lisp-mode-hook
  (lambda ()
    (set (make-local-variable
          lisp-indent-function)
         'common-lisp-indent-function)))
```

For ELI, add the following:

```
(add-hook 'fi:common-lisp-mode-hook
  (lambda ()
    (set (make-local-variable
          lisp-indent-function)
         'common-lisp-indent-function)))
```

- Both ILISP and ELI have commands bound to keys that will allow you to close all parentheses. The ILISP command is close-all-lisp and the ELI one is fi:super-paren

- Both ILISP and ELI have their own completion mechanisms (e.g. – you type “(def” and then press an ILISP/ELI-defined key to get a list of possible completions). Emacs also comes with a completion mechanism that I’ve found to be very useful, even when I haven’t started up a Lisp environment – hippie-expand (a better version of the standard dabbrev-expand function commonly used for expanding abbreviations). I have the following in my .emacs file to setup hippie-expand and bind it to “C-c /”:

```
(require 'hippie-exp)
(setq hippie-expand-try-functions-list
  '(try-expand-dabbrev
    try-expand-dabbrev-all-buffers
    try-expand-dabbrev-from-kill
    try-complete-file-name-partially
    try-complete-file-name
    try-complete-lisp-symbol-partially
    try-complete-lisp-symbol
    try-expand-whole-kill))
(global-set-key [(control c) (/)] 'hippie-expand)
```

This lets me get completion information from the current buffers (which often contain symbols that haven’t been evaluated in the Lisp environment yet), the file system, Lisp symbols, and code that’s been deleted.

- When working with source code, it is sometimes convenient to hide everything except the code you want to modify. The Emacs commands narrow-to-region (“C-x n n”) and narrow-to-defun (“C-x n d”) are useful for doing this. The command widen (“C-x n w”) removes the effect. If you find that you use this a lot, you might want to consider using outline-minor-mode which gives you a number of additional options for working with “outlines” of code.
- When commenting out code, the standard Emacs function comment-dwim (“M-;”) generally does the right thing. It will comment individual lines or regions. For sexp-specific commenting, Paul Foley’s insert-balanced-comments and remove-balanced-comments are very nice [20].
- When making extensive modifications to source code or when working with multiple versions of the same source, the Emacs ediff utility can be very useful for determining where changes have been made. Emacs also has a merge facility that facilitates merging code revisions that have been done by different people to the same source code.

4.3 Evaluating/Compiling Lisp

Typically, when working with Lisp code in Emacs, code is evaluated and/or compiled as it is being worked on. This is normally done from within the buffer of the source file that is being edited, and is often done on a selective basis (e.g. – just a sexp, or just a defun). This allows the programmer to have instant feedback on the code that is being entered.

Each of the Lisp modes has a different range of commands for evaluating and compiling Lisp code. Inferior Lisp Mode has the least number of options (e.g. -- only providing a command to evaluate the last sexp), while ILISP has the most number of options (e.g. -- providing commands to evaluate all changes, the last sexp, the next sexp, the currently selected region, the current defun, or a “Do what I mean” (DWIM) facility that evaluates

based on where the cursor is located). ELI has a similar set of eval/compile commands. For consistency, I've created [21] a DWIM function that works across ILISP, ELI and Emacs Lisp.

4.4 Searching Lisp Code

Locating particular “chunks” of source code can be quite useful when trying to understand how a program works or when trying to make changes to the code. Luckily, Emacs provides many different options for finding and changing source code. In addition, the Lisp modes and add-on packages can provide more specialized searching capabilities. The following subsections deal with these (from the most basic to the more sophisticated).

4.4.1 Standard Emacs Text Search

Emacs has a number of useful standard search functions:

- “C-s” does an incremental search forward (e.g. – as each key is the search string is entered, the source file is searched for the first match. This can make finding specific text much quicker as you only need to type in the unique characters. Repeat searches (using the same search characters) can be done by repeatedly pressing “C-s”
- “C-r” does an incremental search backward
- “C-s RET” and “C-r RET” both do conventional string searches (forward and backward respectively)
- “C-M-s” and “C-M-r” both do regular expression searches (forward and backward respectively)
- “M-%” does a search/replace while “C-M-%” does a regular expression search/replace

All of these are useful but none of the commands have any knowledge of Lisp. When using any of these search commands, it is useful to know that “C-w” will grab the word that is at the cursor and repeatedly pressing “C-w” will extend that to subsequent words.

4.4.2 Finding Occurrences

Emacs provides commands to locate instances of a string (or regular expression) in either the current source file or multiple source files.

- Occur: current source file.
- Grep: multiple files. Note that the Emacs grep command calls out to an external grep. Grep is commonly available on UNIX® machines but is not normally available on Windows. In order to use the Emacs grep command on Windows, it is necessary to either install a 3rd party grep utility [22] or configure Emacs to use the standard Windows findstr utility. This latter approach is done by putting the following line in your .emacs file:

```
(setq grep-command "findstr /n /s ")
```

Again, neither occur nor grep have any knowledge of Lisp.

4.4.3 Finding Lisp Symbols in a Source File

Another standard Emacs utility that can be useful when searching for Lisp code is imenu. It knows about Lisp, so can identify macros, functions, variables, etc. It doesn't require a separate tag generation phase (used by the technique described in the next section) and is useful when working with Lisp symbols in a single source file (whereas the tag files described in the next section can be used to locate symbols across multiple source files).

To activate imenu and to configure it to add a “Symbols” menu to your menu bar when editing Lisp source files, add the following to your .emacs file:

```
(add-hook 'lisp-mode-hook
  (lambda ()
    (imenu-add-to-menubar "Symbols")))
```

I like to have imenu available when I push my middle mouse button, so I also have the following in my .emacs file:

```
(global-set-key [down-mouse-2] 'imenu)
```

Unfortunately, imenu does not work with ELI, so it can only be used if you're using Inferior Lisp Mode or ILISP to work with Lisp code.

4.4.4 Finding Lisp Symbols in Multiple Source Files

Emacs provides a standard Tags utility that knows about Lisp. It can identify and record the location of Lisp variables, functions, macros, classes, methods, etc. However, in order to use it, it is necessary to manually create (and subsequently update) the tag file. This normally requires a series of commands like the following:

```
etags *.lisp
```

to create tags for all Lisp files in the current directory

```
etags -a subdir/*.lisp
```

to add tags to an existing tag file for all the Lisp files in a directory

For example, when I'm working with CLISP (a Common Lisp implementation), I typically like to add all the standard CLISP Lisp source to the tag file that I create for a project; therefore, I execute something like the following (on Windows):

```
c:/bin/emacs/bin/etags.exe c:/bin/clisp/src/*.lisp
```

```
c:/bin/emacs/bin/etags.exe -a *.lisp
```

Once a tag file has been created, it is easy to go directly to the source of any Lisp symbol by simply placing the cursor on the symbol and pressing “M-.” (Note: “C-M-.” is the regular expression version) or to find the next definition by subsequently pressing the “M-,” key sequence. “M-*” returns you back to your original position in the source. If you don't want your original source code to disappear and be replaced by the source code for the definition, you can use “C-x-4-.” to open the definition in another window or “C-x-5-.” to open the definition in another frame. When adding code, the Emacs function complete-tag can be used to complete a partial symbol name based on the tag file tags. The tags-search function is also useful for executing a search in just the files that have been tagged (even though they may span multiple directories).

4.4.5 Finding Lisp Symbols Using Lisp

When Lisp code has been loaded into a Lisp environment, the Lisp interpreter knows a lot about both the symbols that were loaded in and the source code that the symbols originated from. Both ILISP and ELI have extended versions of the Emacs tag facilities that allow you to find the source location of a symbol based on the “knowledge” the Lisp interpreter has about it. The advantage of this approach is that no tags file needs to be built in order to find a definition in source code. The disadvantage is that the code needs to be loaded into the Lisp first (which, when

you're just casually exploring Lisp source code, may not always be the case). Luckily both the ILISP and the ELI functions allow you to "fall back" to tag files if the Lisp image does not contain the definition that is being searched for.

Both ILISP and ELI provide functions that let you view/edit the callers of a function.

4.4.6 Finding Lisp Symbols Using ECB

The ECB (Emacs Code Browser) [23] is an add-on package developed by Eric Ludlum (who also developed EIEIO, a CLOS-like package for Emacs Lisp). For people who like lots of little windows on their desktop, this tool provides a wealth of information about your current project. It is highly configurable and you can easily toggle between using multiple windows and just a source window. One of the windows that you can have displayed shows all the symbols in your current source file. The following figure shows a source file with an ECB buffer on the left side with Lisp symbols in it.

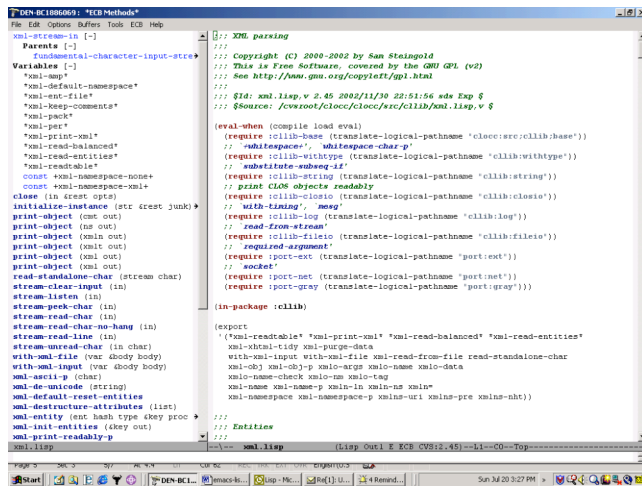


Figure 3. ECB with Methods window on left

Out of the box, ECB doesn't support Common Lisp; however, it can easily be configured to support CL symbols if you use the Emacs Lisp support that is built into ECB for CL programs. The following code snippet shows the basic configuration necessary in order to use ECB with CL:

For Inferior Lisp Mode or ILISP:

```
(add-hook 'lisp-mode-hook
  (lambda ()
    (semantic-default-elisp-setup)))
```

For ELI, add the following:

```
(add-hook 'fi:common-lisp-mode-hook
  (lambda ()
    (semantic-default-elisp-setup)))
```

5. LISP DOCUMENTATION IN EMACS

5.1 Learning About Lisp Symbols

All of the different Lisp development modes have features that allow the user to learn about CL symbols using the built-in capabilities of the Lisp interpreter. However, their implementations of this support differ in completeness and usability.

- Argument lists: Inferior Lisp Mode, ILISP and ELI all have commands that can be evoked to display argument lists on-demand. Both ILISP and ELI can be setup to automatically display the argument list in the minibuffer when a function is typed in – this is similar to the elisp eldoc feature and is quite convenient. ILISP can also insert an arglist template directly into the buffer if you use arglist-lisp with a prefix. This can be convenient for large argument lists which disappear from the minibuffer as soon as you start typing. Similarly, ELI, in addition to showing the arglist in the minibuffer, can be configured to have a separate buffer that shows the argument list.
- Documentation: All Lisp development modes have commands that will show the documentation strings of functions.
- Describe: All Lisp development modes have commands that will describe a symbol.
- Inspect: ILISP has a full-featured inspect function. ELI has support for inspect, but not on the Windows version of ACL. Inferior Lisp Mode does not have an inspect function.
- Macroexpand: Both ILISP and ELI have macroexpand functions. ELI also has a lisp-macroexpand-recursively function which calls cloc::walk-form to macroexpand everything!

5.2 Lisp Documentation

It is useful to have the CL language specification available while you are programming in order to lookup documentation and examples in the spec. Luckily, this documentation is easily (and freely) accessible and it is easy to setup key bindings to retrieve the appropriate section of the manual based on where the cursor is in the source code. The documents that I like to have available locally on my Emacs CL development computer are:

- The Common Lisp HyperSpec: This is the non-official specification for ANSI Common Lisp, developed and made available by Xanalis and Kent Pitman from a very late draft of the ANSI CL specification. It is convenient to have a local HTML copy [24] and an info copy [25] (the info copy is easier to browse). Regardless of whether you use Inferior Lisp Mode, ILISP, or ELI, you should have a copy of this accessible during CL development work. ILISP comes with a handy utility for interactively accessing the appropriate section of the HyperSpec based on the Lisp symbol name the cursor is on. Although Inferior Lisp Mode and ELI don't have any equivalent, you can still download the ILISP utility from CVS [26] and use it. You just need to make certain that it is available in your Emacs load-path and have something similar to the following in your .emacs file (Note: I configure the function key F1 to globally access the HyperSpec, regardless of the mode that I'm in – you may want to limit this to Lisp modes):

;; following 3 are dependent on where you put things

```
(defvar common-lisp-hyperspec-root (concat "c:/home/"
"docs/Hyperspec/"))
```

```
(defvar common-lisp-hyperspec-symbol-table (concat
common-lisp-hyperspec-root "Data/Map_Sym.txt"))
```

```
(defvar hyperspec-prog (concat use-home
"site/ilisp/extra/hyperspec"))
```

```
(global-set-key [f1]
'(lambda ()
(interactive)
(load-library hyperspec-prog)
(common-lisp-hyperspec
(thing-at-point 'symbol))))
```

- CLtL2 (Common Lisp the Language, 2nd Edition): Although CLtL2 refers to an earlier version of CL, much of what is in it is still applicable to ANSI CL. There is a list of identified differences [27] if you want to know how ANSI CL differs from CLtL2. There are hardcopy [28], HTML [29], and info [30] versions of CLtL2 available. Again, ILISP has a function that will allow you to interactively access CLtL2 and it can be downloaded from CVS [31] if ILISP is not being used.
- ACL Documentation: Franz has included with their ACL product CL documentation based on the ANSI CL specification. Their documentation also covers Franz-specific enhancements, so ACL users may find it preferable to access the Franz documentation. Unfortunately, Franz doesn't provide any standard mechanism to access the documentation from ELI; however, Larry Hunter has written an Emacs Lisp package [32] that will access the ACL documentation in a manner similar to the ILISP hyperspec function.

6. USING LISP IN EMACS

6.1 Lisp Listener

When developing CL code in Emacs, the typical development cycle involves working at a toplevel-sexp (e.g. -- defun, defclass, etc.) level of granularity. This often involves working in a combination of the Lisp source and a Lisp listener and is a very dynamic way to develop code. A typical cycle might be:

1. Start up ILISP/ELI
2. "Load" existing Lisp files
3. Open a new Lisp source file or an existing one for modification
4. Add/edit a definition in the source buffer
5. Eval/compile the definition in the source buffer
6. Switch to the listener and test the new/changed definition
7. Repeat steps 4-6
8. Optionally save off a Lisp image with the definitions that have been loaded

This development cycle is much more interactive than the typical C/Java edit/compile/debug development cycle. The ability to dynamically change Lisp code in a running Lisp listener is another feature that is not normally available to C/Java programmers (some Java application servers now provide a "hot fix" feature –

this is something that Lisp developers have been able to do for decades!).

6.2 Project Management

For many projects in different programming languages, "make" is an integral part of the development process. Although the make utility can still be used, it is more common for Lisp developers to use a Lisp-based project management utility. The commercial Lisp products often come with their own utilities for managing Lisp projects; however, there are a number of open source project management utilities that have become increasingly popular. The two that are most commonly used are mk-defsystem [33] and asdf [34].

6.3 Debugging

There is no standard for Lisp debuggers and each implementation tends to have its own specific debugging commands that are typically evaluated in the listener window in Emacs. A debug facility similar to that provided by the Emacs Lisp edebug utility (edebug "instruments" elisp code so that you can visually step through source code when debugging) would be nice to have; however, to date, nobody has developed something like this for Emacs.

ILISP provides a common set of debugging key bindings that work consistently across different implementations and this is advantageous if you routinely switch between Lisp implementations.

7. QUESTIONS/ANSWERS

There are a number of typical problems that developers using Emacs for Lisp development work often encounter. This section discusses those problems in a question/answer format:

- Q1 I get irritated by ELI's switching to an output buffer when I evaluate a sexp in a Lisp source buffer.
- A1 You can control where ELI output goes to by setting the `fi:pop-up-temp-window-behavior` variable. Alternatively, you can use my `copy-eval-dwim-lisp` function [35]. It copies Lisp code from the source buffer to the listener buffer and evaluates it there. Both buffers stay visible and focus remains in the source buffer. The code works for ILISP, ELI and Emacs Lisp.
- Q2 I like having access to the HyperSpec when I'm in Emacs, but why does it have to use an external browser? Why can't I just see the HyperSpec in Emacs?
- A2 If you use the Emacs add-on package W3 (or W3M which provides similar functionality) [36], you can display HTML pages inside of Emacs. Once you have W3 and the HyperSpec both installed, use code similar to the following to access the HyperSpec from the Shift-F1 key:

```
(global-set-key [(shift f1)]
'(lambda ()
(interactive)
(let ((browse-url-browser-function
'browse-url-w3)
(common-lisp-hyperspec-root
"file://c:/home/docs/Hyperspec/"))
```

```
(common-lisp-hyperspec-symbol-table
 (concat common-lisp-hyperspec-root
  "Data/Map_Sym.txt"))

(hyperspec-prog
 "c:/home/site/ilisp/extra/hyperspec")

(load-library hyperspec-prog)

(common-lisp-hyperspec
 (thing-at-point 'symbol))))
```

Note that the “let” in the above code sets the browse-url-browser-function to W3 for just the HyperSpec. You can either set the variable globally (if you want to always use W3 or some other specific browser) or locally (if you want to use a specific browser and not the default one).

- Q3 I switch between UNIX® and Windows environments and, although Emacs makes this switch a lot easier, I find it inconvenient having to use different Shell environments on different operating systems.
- A3 On Windows, the cygwin [37] tools provide a lot of the same tools that are available under UNIX® as well as a BASH shell. Alternatively, you might want to consider using eshell [38], a shell written in Emacs Lisp that comes as a standard feature in later releases of Emacs.
- Q4 I would like to use Emacs with Franz’s ACL but find that I use the Franz tools so much that I can’t afford to not load their IDE.
- A4 It doesn’t have to be an either/or decision. On Windows, Franz allows you to specify (under Options) that Emacs is to be the default editor in place of their built-in editor. On UNIX®, Emacs also works very well together with the Franz tools.
- Q5 I want to use Emacs on a Windows machine. Unfortunately, I have the Windows cut/copy/paste key bindings burned into my fingertips and would find it very difficult to switch back and forth between the Windows standard for these shortcut keys and the Emacs standard.
- A5 Luckily, you don’t have to! Download cua.el [39] and you can continue to use the Windows defaults. In fact, you may find that the following commands in your .emacs file will make Emacs more Windows-like:
- ```
:: Windows-like mouse/arrow movement & selection
(pc-selection-mode)
(delete-selection-mode t)
:: C-z=Undo, C-c=Copy, C-x=Cut, C-v=Paste (needs cua.el)
(require 'cua)
(CUA-mode t)
```
- Q6 There was a lot of Emacs Lisp code presented in this paper. Do I really have to type in all this stuff to get started with Emacs and Lisp?
- A6 No, there is a sample .emacs file [40] that can be used to get started. It contains all of the configurations that have been described in this paper and (hopefully) should work with some minor tweaking.
- Q7 I’ve tried out Emacs and I just can’t get used to it. What other Lisp-friendly alternative are there?

A7 The Franz [41], Xanalys [42], Corman [43], and Digitool [44] commercial Lisp offerings all have Lisp-aware editors.

CMUCL has Hemlock [45], which is also being adapted for other Lisps [46].

XEmacs [47] is an alternative to GNU Emacs that works with many of the same Elisp libraries. Some people prefer it to GNU Emacs.

Vim can be used to edit Lisp code. An article [48] by Larry Clapp gives some pointers on how to use Vim with Lisp.

Jabberwocky [49] is a Lisp editor/debugger written in Java.

Lastly, for true masochists, notepad on Windows or ed on UNIX® can also be used. :-)

Q8 Where can I get more information and updates to this paper?

A8 A version of the material in this paper will eventually be available on the CL Cookbook site [50]. That information will be updated periodically. In the short term, updates and supplemental material will also be available at my ILC2003 page. [51].

## 8. ACKNOWLEDGMENTS

My thanks to Edi Weitz, Jans Aasman, Pascal Costanza, Peter Santoro, David Steuber, Andre Van Meulebrouck and Kevin Layer (Franz) for reviewing and providing feedback on initial drafts of this paper.

## 9. REFERENCES

- [1] GNU Emacs can be downloaded from:  
<http://www.gnu.org/software/emacs/emacs.html#Obtaining>
- [2] Emacs on CD can be ordered from:  
<https://agia.fsf.org/>
- [3] I recommend the following introductory Emacs books:
- Ayers, Larry, Gnu Emacs and XEmacs, Premier Press, 2001  
<http://www.amazon.com/exec/obidos/tg/detail/-/0761524460>
- Cameron, Debra et al, Learning Gnu Emacs, 2<sup>nd</sup> Edition, O’Reilly & Associates, 1996  
<http://www.oreilly.com/catalog/gnu2/>
- [4] Bob Rogers’ Emacs page:  
<http://rgrjr.dyndns.org/emacs/>
- [5] Emacs Wiki:  
<http://www.emacswiki.org/cgi-bin/wiki.pl>
- [6] EIEIO – A CLOS-like package for Emacs Lisp:  
<http://cedet.sourceforge.net/eieio.shtml>
- [7] For learning Emacs Lisp, I recommend the following:  
An Introduction to Programming in Emacs Lisp is available in hard-copy or a number of different electronic formats:  
<http://www.gnu.org/manual/emacs-lisp-intro/emacs-lisp-intro.html>
- [8] The following is also a good book on Emacs Lisp:  
Glickstein, Bob, Writing GNU Emacs Extensions, O’Reilly & Associates, 1997  
<http://www.oreilly.com/catalog/gnuext/>

- [9] Early keyboards were an influence on Emacs:  
<http://world.std.com/~jdostale/kbd/>
- [10] Alternative Emacs-friendly keyboards:
- Happy Hacking keyboard:  
<http://store.yahoo.com/pfuca-store/>
- Kinesis keyboards:  
<http://www.kinesis-ergo.com/>
- FingerWorks TouchStream keyboard:  
[http://www.fingerworks.com/lp\\_product.html](http://www.fingerworks.com/lp_product.html)
- [11] Swapping keys on UNIX®:  
<http://rgrjr.dyndns.org/emacs/keyboard.html>
- Swapping keys on Mac OS X:  
<http://gnufoo.org/ucontrol/ucontrol.html>
- Swapping keys on Windows:  
Manually:  
<http://www.microsoft.com/whdc/hwdev/tech/input/w2kscan-map.mspcx>
- Using a utility:  
<http://www.cam.hi-ho.ne.jp/oishi/indexen.html>
- [12] Foot pedals that can be used with Emacs:  
<http://www.kinesis-ergo.com/>
- [13] Common Lisp HyperSpec:  
<http://www.lispworks.com/reference/HyperSpec/>
- [14] Common Lisp the Language:  
<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/cltl/cltl2.html>
- [15] ILISP:  
<http://sourceforge.net/projects/ilisp/>
- [16] ILISP installation instructions are available on the CL Cookbook site for Windows:  
<http://cl-cookbook.sourceforge.net/windows.html>
- or Dan Barlow's instructions for UNIX®:  
<http://ww.telent.net/lisp/howto.html>
- [17] ELI port to Sciener:  
<http://www.sciener.com/sc/>
- [18] ELI port to CMUCL:  
<http://www.cons.org/cmucl/doc/eli-hacks.txt>
- [19] ELI port to SBCL:  
<http://lemonodor.com/archives/000461.html>
- [20] Paul Foley's insert-balanced-comments and remove-balanced-comments functions:  
<http://users.actrix.co.nz/mycroft/comments.el>
- [21] My copy-eval-dwim-lisp function (and the supporting copy-eval-last-sexp function) is available at:  
[http://home.comcast.net/~b.clementson/ilc\\_2003.htm](http://home.comcast.net/~b.clementson/ilc_2003.htm)
- [22] Google can be used to find a number of different stand-alone grep utilities for use with Windows. The cygwin utilities contain one as well:  
<http://www.cygwin.com/>
- [23] The Emacs Code Browser (ECB) is available from:  
<http://ecb.sourceforge.net/>
- [24] Common Lisp HyperSpec in HTML format:  
[ftp://ftp.xanalys.com/pub/software\\_tools/reference/HyperSpec-6-0.tar.gz](ftp://ftp.xanalys.com/pub/software_tools/reference/HyperSpec-6-0.tar.gz)
- [25] Common Lisp HyperSpec in info format:  
<ftp://ftp.gnu.org/pub/gnu/gcl/gcl.info.tgz>
- [26] The hyperspec.el ILISP program in CVS:  
<http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ilisp/ILISP/extra/hyperspec.el>
- [27] Differences between the CL described in CLtL2 and the ANSI CL definition of CL:  
<http://home.comcast.net/~bc19191/cltl2-ansi.htm>
- [28] CLtL2 – hard-copy format:  
<http://www.amazon.com/exec/obidos/tg/detail/-/1555580416>
- [29] CLtL2 – HTML format:  
[http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/lang/lisp/doc/cltl/cltl\\_ht.tgz](http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/lang/lisp/doc/cltl/cltl_ht.tgz)
- [30] CLtL2 – Info format:  
<http://www.sfu.ca/~sabetts/docs/>
- [31] The cltl2.el ILISP program in CVS:  
<http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ilisp/ILISP/extra/cltl2.el>
- [32] Larry Hunter's aelc.el program:  
<http://compbio.uchsc.edu/Hunter/aelc.el>
- [33] The mk-defsystem utility is part of CLOCC:  
<http://sourceforge.net/projects/clocc>
- [34] The asdf utility is available from the cCLan project:  
<http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/cclan/asdf/>
- [35] My copy-eval-dwim-lisp function (and the supporting copy-eval-last-sexp function) is available at:  
[http://home.comcast.net/~b.clementson/ilc\\_2003.htm](http://home.comcast.net/~b.clementson/ilc_2003.htm)
- [36] The W3 Internet Browser for Emacs:  
<http://www.cs.indiana.edu/elisp/w3/docs.html>
- The W3M Internet Browser for Emacs:  
<http://emacs-w3m.namazu.org/>
- [37] The cygwin tools:  
<http://www.cygwin.com/>
- [38] Eshell is part of Emacs; however, documentation is only available in the source code. Some online documentation is available at:  
<http://www.emacswiki.org/cgi-bin/wiki.pl?CategoryEshell>
- [39] The cua.el utility program:  
<http://www.emacswiki.org/cgi-bin/wiki.pl?CuaMode>
- [40] The sample .emacs file:  
[http://home.comcast.net/~b.clementson/ilc\\_2003.htm](http://home.comcast.net/~b.clementson/ilc_2003.htm)
- [41] The Franz ACL web site:  
<http://www.franz.com/>

- [42] The Xanalys LispWorks web site:  
<http://www.lispworks.com/>
- [43] The Corman web site:  
<http://www.cormanlisp.com/>
- [44] The Dgitool web site:  
<http://www.dgitool.com/>
- [45] CMUCL's Hemlock editor:  
<http://www.cons.org/cmucl/hemlock/index.html>
- [46] Portable Hemlock:  
<http://www.stud.uni-karlsruhe.de/~unk6/hemlock/>
- [47] XEmacs:  
<http://www.xemacs.org/>
- [48] "Lisp with Vim" article by Larry Clapp:  
<http://lisp-p.org/15-vim/>
- [49] The Jabberwocky editor:  
<http://jabberwocky.sourceforge.net/>
- [50] The CL Cookbook:  
<http://cl-cookbook.sourceforge.net/>
- [51] Updates and additional material related to this paper:  
[http://home.comcast.net/~b.clementson/ilc\\_2003.htm](http://home.comcast.net/~b.clementson/ilc_2003.htm)