

# ILISP User Manual

---

A GNU Emacs Interface for Interacting with Lisp  
Edition 0.24, December 2002  
For ILISP Version: 5.12.1

by Todd Kaufmann, Chris McConnell, Ivan Vazquez,  
Marco Antoniotti, Rick Campbell  
and Paolo Amoroso

---

Copyright ©

- 1991, 1992, 1993 Todd Kaufmann
- 1993, 1994 Ivan Vasquez
- 1994, 1995, 1996 Marco Antoniotti and Rick Busdiecker
- 1996, 1997, 1998, 1999 Marco Antoniotti and Rick Campbell

This is edition 0.24 of the *ILISP User Manual* for ILISP Version: 5.12.1, December 2002.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by this author.

## How to get the latest ILISP distribution.

ILISP is “free”; this means that everyone is free to use it and free to redistribute it on a free basis. ILISP is not in the public domain; it is copyrighted and there are restrictions on its distribution, but these restrictions are designed to permit everything that a good cooperating citizen would want to do. What is not allowed is to try to prevent others from further sharing any version of ILISP that they might get from you. The precise conditions appear in the file ‘COPYING’.

The easiest way to get a copy of ILISP is from someone else who has it. You need not ask for permission to do so, or tell any one else; just copy it.

General information on ILISP is available at:

<http://ilisp.cons.org/>

The project site, which provides access to the CVS source tree, bug database, mailing lists and other resources, is hosted at SourceForge:

<http://sourceforge.net/projects/ilisp/>

The following mailing lists are available:

‘ilisp-announce’

Subscribe to this list if you want to receive public announcements regarding ILISP.

‘ilisp-devel’

This is the list for people who want to be actively involved in the development of ILISP.

‘ilisp-help’

This is the list for asking usage questions regarding ILISP.

‘ilisp-cvs’

Subscribe to this list *only* if you want to monitor the CVS repository activity.

You can subscribe to the lists and access the archives via the general information pages:

- <http://lists.sourceforge.net/mailman/listinfo/ilisp-announce>
- <http://lists.sourceforge.net/mailman/listinfo/ilisp-devel>
- <http://lists.sourceforge.net/mailman/listinfo/ilisp-help>
- <http://lists.sourceforge.net/mailman/listinfo/ilisp-cvs>

You may send bug reports, questions, suggestions, etc. to ‘ilisp-help’ or ‘ilisp-devel’. To report a bug you can also switch to the buffer where the problem occurs, execute the M-x `ilisp-bug` command and follow the displayed instructions.

## FTP and Web directions

You can get the distribution file, `ilisp-x.y.z.tar.gz` (`ilisp-x.y.z.zip`), via HTTP or anonymous FTP at the following addresses:

- [http://sourceforge.net/project/showfiles.php?group\\_id=3957](http://sourceforge.net/project/showfiles.php?group_id=3957)
- <http://www2.cons.org:8000/ftp-area/ilisp/>

- `ftp://ftp2.cons.org/pub/languages/lisp/ilisp/`

If you use a tty ftp client, just log in as 'anonymous'.

Please report any problems to the 'ilisp-help' mailing list. See [Distribution], page 1.

Unpack and install:

```
% gzip -dc ilisp-x.y.z.tar.gz | tar xf -
```

or

```
% unzip ilisp-x.y.z.zip
```

See Chapter 1 [Installation], page 7.

## Acknowledgements

ILISP replaces the standard inferior Lisp mode. ILISP is based on comint mode and derived from a number of different interfaces including Symbolics, CMU Common Lisp, and Thinking Machines.

There are many people that have taken the time to report bugs, make suggestions and even better send code to fix bugs or implement new features.

Thanks to Paolo Amoroso, Martin Atzmueller, David Bakash, Daniel Barlow, Guido Bosch, Tim Bradshaw, David Braunegg, Thomas M. Breuel, Craig Brozefsky, Rick Campbell, Hans Chalupsky, Bill Clementson, Kimball Collins, William Deakin, Brian Dennis, David Duff, Tom Emerson, Michael Ernst, Scott Fahlman, Karl Fogel, Dave Fox, Paul Fuqua (for the CMU-CL GC display code), David Gadbois, Robert P. Goldman, Marty Hall, Richard Harris, Utz-Uwe Haus, Jim Healy, Matthias Hoelzl, Christopher Hoover, Larry Hunter, Ben Hyde, Chuck Irvine, Mark Kantrowitz, Michael Kashket, Matthias Koeppel, Hannu Koivisto, Qiegang Long, Christian Lynbeck, Erik Naggum, Alain Picard Dan Pierson, Yusuf Pisan, Frank Ritter, Ole Rohne, Kevin Rosenberg, Jeffrey Mark Siskind, Neil Smithline, Richard Stallman, Larry Stead, Jason Trenouth, Christof Ullwer, Reini Urban, Bjorn Victor, Edmund Weitz, Fred White, Ben Wing, Matsuo Yoshihiro, Jamie Zawinski and many others for bug reports, suggestions and code. Our apologies to anyone we may have forgotten.

Special thanks to Todd Kaufmann for the texinfo file, work on bridge, epoch-pop and for really exercising everything.

Please send bug reports, fixes and extensions to the ‘ilisp-devel’ mailing list so that they can be merged into the master source. See [Distribution], page 1.

--Chris McConnell	1991-03-18
--Ivan Vazquez	1993-06-27
--Marco Antoniotti and Rick Campbell	1996-10-25
--Marco Antoniotti and Paolo Amoroso	1999-08-19



## Features

ILISP is an interface from GNU Emacs to an inferior Lisp. It has the following features:

- Runs under Emacs 18 through 21, and XEmacs 19 through 21.
- Support for multiple Common Lisp (including Allegro, CLISP and CMU), XLisp and Scheme dialects on multiple machines even at the same time.
- Dynamically sized pop-up windows that can be buried and scrolled 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.
- 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 or compilation. The user can also specify that a function definition should be called after evaluation. 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.
- Completion of filename components and Lisp symbols including partial matches.
- 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.
- 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 of the current “defun” (in Emacs terminology).
- Handles editing, entering and indenting full Lisp expressions.
- Next, previous, and similar history mechanism compatible with comint.
- Handles Lisp errors.
- Uniform interface to Lisp debuggers.
- Result histories are maintained in the inferior Lisp.
- Does not create spurious symbols and handles case issues.
- Online manuals for ILISP and Common Lisp.





# 1 How to install ILISP

Installation of ILISP and some initialization of your computing environment are described in this chapter. Please read the following sections carefully before getting started with ILISP.

Copy the ILISP distribution archive, e.g. `ilisp-x.y.z.tar.gz`, to the location where you would like to install it. Next extract the archive, see See [FTP and Web directions], page 1. You may need root privileges to perform these operations.

## 1.1 Configuration and compilation

Some configuration needs to be done before compiling the Emacs Lisp files that comprise ILISP. Start with the `Makefile` file, in the section after the comment **Various variables** (you can safely ignore the variables for configuring packaging and distribution, which are intended for maintainers).

First, set the `EMACS` variable to be the pathname of the Emacs you will be using ILISP with. This is the Emacs that will be used to compile ILISP with. Be sure to set `LN` to the name of your operating system's command for creating symbolic filesystem links, **especially if you are a Windows user**.

If your Emacs supports the `easymenu` package, it is possible to make ILISP add to Lisp mode buffers and buffers with inferior Lisp processes, or to Scheme mode buffers and buffers with inferior Scheme processes, an `Ilisp` menu with all available commands. To enable this feature, set to `t` the variable `ilisp-*enable-cl-easy-menu-p*` in `'ilisp-def.el'` for the Common Lisp dialects, and `ilisp-*enable-scheme-easy-menu-p*` for Scheme dialects. Setting these variables also causes the default `Lisp` menu to be removed before displaying the `Ilisp` one.

See the file `'INSTALLATION'` for additional configuration options and known problems for specific Lisp dialects.

Run `make` or `make compile` to build ILISP from source. Ignore any compilation warnings unless they result in ILISP not compiling completely. If you are a Windows user, and you don't have GNU `make`, you can still compile ILISP by running the `'icompile.bat'` batch file (be sure to customize for your system the variables mentioned by the comment at the top).

For reducing the Emacs startup time you may run `make loadfile`. This concatenates all `'*.elc'` (the compiled Emacs Lisp files) into an `'ilisp-all.elc'` file and removes the `'*.elc'` files. So your Emacs can load one single compiled file faster than a bunch of smaller compiled files.

To activate ILISP you should add appropriate Emacs Lisp forms to your `'*.emacs'` or to the system-wide `'default.el'` file, depending on who will be using ILISP. These forms take care of starting it whenever you access a Lisp file or run an inferior Lisp process. You can copy relevant portions of the sample file `'ilisp.emacs'`, which also shows how to customize some ILISP features.

You should add the directory where all of the ILISP Emacs Lisp files reside to your `load-path`. There is an example of this in `'ilisp.emacs'`.

As an alternative you could set up a `‘.ilisp’` which contains the appropriate portions of `‘ilisp.emacs’`, in order to avoid cluttering too much `‘.emacs’` or `‘default.el’`.

The first time a dialect is started, the interface files will complain about not being compiled, just ignore the message. Once a Lisp dialect is started up, you should execute the command `ilisp-compile-inits` which will compile the `‘*.lisp’` files and write them to the same directory as the ILISP files.

The binary files should have a unique extension for each different combination of architecture and Lisp dialect. You will need to change `ilisp-init-binary-extension` and `ilisp-init-binary-command` to get additional extensions. The binary for each different architecture should be different. If you want to build the interface files into a Lisp world, you will also need to set `ilisp-load-inits` to `nil` in the same place that you change `ilisp-program` to load the Lisp world.

There is an `ilisp-site-hook` for initializing site specific stuff like program locations when ILISP is first loaded. You may want to define appropriate autoloads in your system Emacs start up file.

Example site init:

```
;;; CMU site
(setq ilisp-site-hook
  '(lambda ()
    (setq ilisp-motd "CMU ILISP V%s")
    (setq expand-symlinks-rfs-exists t)
    (setq allegro-program "/usr/local/acl5/lisp")
    (setq lucid-program "/usr/misc/.lucid/bin/lisp"))))
```

Kent Pitman and Xanalis Inc. have made publicly available on the Web the Common Lisp HyperSpec, an HTML version of the full text of the ANSI Common Lisp specification:

[http://www.xanalis.com/software\\_tools/reference/HyperSpec/](http://www.xanalis.com/software_tools/reference/HyperSpec/)

It is also possible to get a local copy of the HyperSpec, whose latest version is currently v6, by downloading the file `HyperSpec-6-0.tar.gz` from the above mentioned site.

Daniel Barlow, Stephen Carney and Erik Naggum independently developed Emacs Lisp packages for looking up Lisp symbols in the HyperSpec and displaying the relevant sections with a Web browser. ILISP used to include all of them in the `‘extra’` directory of the distribution tree. However, because of some changes to the CLHS only Erik Naggum’s version is now distributed. If you want to use one of the others, please contact the other authors.

The `‘ilisp.emacs’` file provides sample instructions for making Naggum’s package access a local copy of the HyperSpec. Since the package relies on the `browse-url` Emacs package, make sure that the latter is properly configured.

Digital Press has made publicly available online, as a service to the Lisp community, the full text of the book “Common Lisp, The Language” (by Guy L. Steele Jr., 2nd edition, Digital Press, 1990, ISBN 1-55558-041-6; a.k.a. “CLtL2”) in a number of formats, including HTML. ILISP provides support, contributed by Utz-Uwe Haus, for looking up Lisp symbols in the HTML version of the book and displaying the relevant sections with a Web browser. See the file `‘extra/cltl2.el’` for more information on configuring this feature. See Section 5.2 [Documentation functions], page 18, for usage instructions.

The `'ilisp.emacs'` file provides sample instructions for making ILISP's CLtL2 support access a local copy of the book. What has been said above about `browse-url` configuration also applies to CLtL2 lookup.

Note that, although Steele's book is a well written and useful resource, it covers the Common Lisp language in the state it was a few years before ANSI standardization. If you need an accurate description of ANSI Common Lisp, see the above mentioned HyperSpec instead.

Previous versions of ILISP provided commands for accessing the online Common Lisp documentation shipped with Franz Inc.'s Allegro CL product (`fi:clman` module). The public availability of the HyperSpec, and the inclusion since version 5.9 of ILISP of the `hyperspec` packages, make access to the Franz documentation no longer necessary. So by default ILISP does not load the `fi:clman` module, but if you still want to use its commands set the `ilisp-*use-fi-clman-interface-p*` to `t` in `'ilisp-def.el'`.

The ILISP documentation consists of a user manual and a reference card (the latter may not be up to date). Both of them are in the `'docs'` directory of the distribution tree.

The generation of GNU Info, DVI, PostScript and HTML versions of the documentation from the Texinfo and TeX source is controlled by the `'Makefile'` in the `'docs'` directory. Run `make docs` or just `make` to generate all of the formats. If you are interested in only some of them then issue the appropriate command: `make info` for GNU Info, `make dvi` for DVI, `make ps` for PostScript and `make html` for HTML. To remove the intermediate files produced during the generation of DVI output you can run `make tmpclean`. Note that some of the output formats may not be supported for certain documents.

The ILISP reference card is available as a TeX source file. Check the comments at the beginning of the file if you need to generate a version with a different number of columns (the default is 3).



## 2 How to run a Lisp process using ILISP

To start a Lisp use *M-x run-ilisp*, or a specific dialect like *M-x allegro*. If one of these two functions is called with a numerical prefix, the user will be prompted for a buffer name and a program to run. The default buffer name is the name of the dialect with \*s around it. The default program for a dialect will be the value of DIALECT-program or the value of ilisp-program inherited from a less specific dialect. If there are multiple Lisp's, use the dialect name or *M-x select-ilisp* ( $\overline{C-Z}$  *S*) to select the current ILISP buffer.

Entry into ILISP mode runs the hooks on `comint-mode-hook` and `ilisp-mode-hook` and then DIALECT-hooks specific to Lisp dialects in the nesting order above. Many dialects call `ilisp-load-init` in their dialect setup.

These are the currently supported dialects.

- `allegro` Allegro Common Lisp from Franz Inc.
- `akcl` Austin Kyoto Common Lisp, the U. Texas derivative.
- `chez` Chez Scheme by Cadence Research Systems.
- `clisp-hs` CLISP by Haible and Stoll.
- `cmulisp` CMU Common Lisp, the major development platform for ILISP so far.
- `cormanlisp` Corman Common Lisp by Roger Corman.
- `drscheme-jr` DrScheme-jr by Rice University's PLT.
- `common-lisp` Generic Common Lisp.
- `ec1` EcoLisp, the Embeddable Common Lisp by Beppe Attardi. A derivative of KCL and AKCL.
- `gcl` GNU Common Lisp, the official GNU release. A derivative of AKCL.
- `guile` GUILE Scheme by the GNU Project.
- `ibcl` Ibuki Common Lisp, derived from KCL.
- `kcl` Kyoto Common Lisp, original version.
- `liquid` Liquid Common Lisp, the successor of Lucid Common Lisp supported by Xanalys/Harlequin Ltd.
- `lispworks` LispWorks Common Lisp from Xanalys/Harlequin Ltd.
- `lucid` Lucid Common Lisp, currently supported by Xanalys/Harlequin Ltd.
- `mzscheme` MzScheme by Rice University's PLT.
- `oaklisp` Oaklisp scheme.
- `openmcl` OpenMCL.
- `sbc1` Steel Bank Common Lisp
- `scheme` Generic Scheme.
- `scm` SCM Scheme by Aubrey Jeffer.
- `snow` Snow, STk Scheme without supoort for the Tk toolkit.
- `stk` STk scheme by Erick Gallesio.
- `xlisp` XLisp by David Betz.
- `xlispstat` XLisp-Stat, a derivative of XLisp for statistical computations.

*Support for Scheme and XLisp dialects is experimental* and your feedback is welcome. The ‘ilisp-s2c.el’ file contains a first cut at defining the Scheme->C dialect, but it is neither compiled nor loaded by ILISP.

To define a new dialect, See Section 7.1 [Defining new dialects], page 31, and See Chapter 6 [Customization], page 29. If anyone figures out support for other dialects, I would be happy to include it in future releases. See Chapter 7 [Dialects], page 31.

The currently supported dialects are listed below so that the indentation corresponds to the hierarchical relationship between dialects:

```
common-lisp
  allegro
  clisp-hs
  cmulisp
  cormanlisp
  kcl
    akcl
      gcl
      ecl
    ibcl
  lispworks
  lucid
    liquid
  openmcl
  sbcl
scheme
  chez
  guile
  mzscheme
    drscheme-jr
  oaklisp
  Scheme->C (still "in fieri")
  scm
  snow
  stk
xlisp
  xlispstat
```

### 3 A word about the keys used by ILISP

By default, most ILISP commands are bound under the prefix key  $\overline{C-z}$ . Unfortunately, these bindings predate the modern FSF Emacs keyspace policies, which stipulate that packages should use  $\overline{C-c}$  as a prefix, and bind only control characters, digits, and a few specific punctuation chars under that prefix.

If you are already accustomed to the old ILISP bindings, don't worry – we haven't changed the default. However, for new users who don't have old habits to unlearn, ILISP offers FSF-compliant bindings as an alternative to the default. To be compliant (and who wouldn't want to be compliant?), put this line in your '.emacs' or in the system-wide 'default.el' file:

```
(setq ilisp-*use-fsf-compliant-keybindings* t)
```

This will cause the ILISP prefix key to be  $\overline{C-c}$ , and also change some of the bindings underneath that prefix. After you do this, ILISP will be FSF-compliant.

Because the rest of this document was originally written for the old, default ILISP bindings, you'll need to make some mental translations if you choose FSF-compliance:

- All key bindings are compliant with the FSF guidelines for major mode key bindings. In some cases, there is an overlap with a key binding that is defined in either standard Emacs or comint mode; however, in these cases, the overlap is consistent with the FSF guideline that "it is reasonable for a major mode to rebind a key sequence with a standard meaning, if it implements a command that does 'the same job' in a way that fits the major mode better".
- Where possible, mnemonic key bindings have been made to facilitate ease of learning and frequently-used commands have been assigned to a key binding that is easy to reach.
- All of the <query> related functions are set up on on  $\overline{C-c C-q}$   $C-<letter>$  key bindings. Rationale: The "q" binding is a mnemonic binding.
- All of the <eval> related functions are set up on on  $\overline{C-c C-j}$   $C-<letter>$  key bindings. Rationale:  $C-c C-j$  is an easy control key sequence to hit and the "eval" functions are used quite a lot.
- All of the <compile> related functions are set up on on  $\overline{C-c C-k}$   $C-<letter>$  key bindings. Rationale:  $C-k$  is mnemonic for "kompile".
- All of the <action>-and-go-lisp functions are of the form  $\overline{C-c C-<character>}$   $M-<letter>$ . The corresponding <action> functions are set up on the  $\overline{C-c C-<character>}$   $C-<letter>$  binding. Rationale: This is an easy sequence to remember - the "and-go-lisp" function is just the exact same sequence as the ordinary function but with the last character modified with Meta rather than control.
- All of the <buffer/edit> related functions are set up on on  $\overline{C-c C-v}$   $C-<letter>$  key bindings. Rationale:  $C-c C-v$  is an easy control key sequence to hit and the "buffer/edit" functions are used quite a lot.
- All of the <debug> related functions are set up on on  $\overline{C-c C-b}$   $C-<letter>$  key bindings. Rationale: These are de"b"ugging commands.

Remember that you can type  $C-h m$  at any time to see help on the current major mode, which will show (among other things) a list of all currently active keybindings.





## 4 Buffers used by ILISP, and their commands

### ***\*dialect\****

The Lisp listener buffer. Forms can be entered in this buffer in, and they will be sent to Lisp when you hit return if the form is complete. This buffer is in `ilisp-mode`, which is built on top of `comint-mode`, and all `comint` commands such as history mechanism and job control are available.

### ***lisp-mode-buffers***

A buffer is assumed to contain Lisp source code if its major mode is in the list `lisp-source-modes`. If it's loaded into a buffer that is in one of these major modes, it's considered a Lisp source file by `find-file-lisp`, `load-file-lisp` and `compile-file-lisp`. Used by these commands to determine defaults.

### ***\*Completions\****

Used for listing completions of symbols or files by the completion commands. See Section 5.13 [Completion], page 26.

### ***\*Aborted Commands\****

See Section 5.10 [Interrupts], page 23.

### ***\*Errors\****

### ***\*Output\****

### ***\*Error Output\****

used to pop-up results and errors from the inferior Lisp.

### ***\*ilisp-send\****

Buffer containing the last form sent to the inferior Lisp.

### ***\*Edit-Definitions\****

### ***\*All-Callers\****

See Section 5.6 [Source code commands], page 20.

### ***\*Last-Changes\****

### ***\*Changed-Definitions\****

See Section 5.7 [Batch commands], page 22.

### ***\*Arglist-Output\****

A buffer for showing arglist messages.

## 4.1 Typeout windows

All ILISP output is funneled through the function which is bound to the hook `ilisp-display-output-function`. The function gets a single argument, a string, and should make that output visible to the user somehow.


One possible choice for output display is `ilisp-display-output-in-typeout-window`, which pops up a window at the top of the current screen which is just large enough to display the output. This window can be “remotely controlled” by the commands `ilisp-scroll-output`, `ilisp-bury-output`, and `ilisp-grow-output`.

Unlike the old popper facility, the `ilisp` typeout window facility does not trounce on any existing Emacs functions or on any common key bindings, like `C-x o`.


Other built-in functions which might be useful as values for `ilisp-display-output-function` include `ilisp-display-output-default`, `ilisp-display-output-adaptively`, `ilisp-display-output-in-lisp-listener`, `ilisp-display-output-in-temp-buffer`, and `ilisp-display-output-in-typeout-window`.

The default display function is `ilisp-display-output-default`, which obeys the `lisp-no-popper` variable.


Users are encouraged to write their own output display functions to get the exact desired behavior, displaying on a private Emacs screen, in a pop-up dialog box, or whatever.

 *1 (ilisp-bury-output)*

deletes and buries the typeout output window.

 *v (ilisp-scroll-output)*

scrolls the output window if it is showing, otherwise does nothing. If it is called with a negative prefix, it will scroll backwards.


 *G (ilisp-grow-output)*

will grow the output window if showing by the prefix number of lines.

An alternative to typeout windows is to always have the inferior Lisp buffer visible and have all output go there. Setting `lisp-no-popper` to `t` will cause all output to go to the inferior Lisp buffer. Setting `lisp-no-popper` to `'message` will make output of one line go to the message window. Setting `comint-always-scroll` to `t` will cause process output to always be visible. If a command gets an error, you will be left in the break loop.

## 4.2 Switching buffers

Commands to make switching between buffers easier.

 *b (switch-to-lisp)*

will pop to the current ILISP buffer or if already in an ILISP buffer, it will return to the buffer that last switched to an ILISP buffer. With a prefix, it will also go to the end of the buffer. If you do not want it to pop, set `pop-up-windows` to `nil`.

*M-C-1 (previous-buffer-lisp)*

will switch to the last visited buffer in the current window or the Nth previous buffer with a prefix.

## 5 ILISP Commands

Most of these key bindings work in both Lisp Mode and ILISP mode. There are a few additional and-go bindings found in Lisp Mode.

### 5.1 Eval and compile functions

In Lisp, the major unit of interest is a form, which is anything between two matching parentheses. Some of the commands here also refer to “defun,” which is a list that starts at the left margin in a Lisp buffer, or after a prompt in the ILISP buffer. These commands refer to the “defun” that contains the point.

“A call” refers to a reference to a function call for a function or macro, or a reference to a variable. Commands which “insert a call” in the ILISP buffer will bring up the last command which matches it or else will insert a template for a call.

When an eval is done of a single form matching `ilisp-defvar-regex` the corresponding symbol will be unbound and the value assigned again.

When you send a form to Lisp, the status light will reflect the progress of the command. In a Lisp mode buffer the light will reflect the status of the currently selected inferior Lisp unless `lisp-show-status` is nil. The very first inferior Lisp command executed may send some forms to initialize the inferior Lisp. If you want to find out what command is currently running, use the command `(C-z) s` (`status-lisp`). If you call it with a prefix, the pending commands will be displayed as well.

Note that in this table as elsewhere, the key `(C-z)` (`ilisp-*prefix*`) is used as a prefix character for ILISP commands, though this may be changed. For a full list of key-bindings, use `M-x describe-mode` or `M-x describe-bindings` while in an ILISP-mode buffer.

The eval/compile commands verify that their expressions are balanced and then send the form to the inferior Lisp. If called with a positive prefix, the result of the operation will be inserted into the buffer after the form that was just sent.

For commands which operate on a region, the result of the compile or eval is the last form in the region.

The ‘`and-go`’ versions will perform the operation and then immediately switch to the ILISP buffer where you will see the results of executing your form. If `eval-defun-and-go-lisp` or `compile-defun-and-go-lisp` is called with a prefix, a call for the form will be inserted as well.

`(C-z)`      The prefix-key for most ILISP commands. This can be changed by setting the variable `ilisp-*prefix*`.

**`RET`** (`return-ilisp`)

In ILISP-mode buffer, sends the current form to lisp if complete, otherwise creates a new line and indents. If you edit old input, the input will be copied to the end of the buffer first and then sent.

**`C-]`** (`close-and-send-lisp`)

Closes the current sexp, indents it, and then sends it to the current inferior Lisp.

*LFD (newline-and-indent-lisp)*

Insert a new line and then indent to the appropriate level. If called at the end of the inferior Lisp buffer and an sexp, the sexp will be sent to the inferior Lisp without a trailing newline.

$\overline{C-Z}$  *e (eval-defun-lisp)*

*M-C-x (eval-defun-lisp)*

$\overline{C-Z}$  *C-e (eval-defun-and-go-lisp)*

Send the defun to Lisp.

$\overline{C-Z}$  *r (eval-region-lisp)*

$\overline{C-Z}$  *C-r (eval-region-and-go-lisp)*

$\overline{C-Z}$  *n (eval-next-sexp-lisp)*

$\overline{C-Z}$  *C-n (eval-next-sexp-and-go-lisp)*

$\overline{C-Z}$  *o (eval-last-sexp-lisp)*

$\overline{C-Z}$  *C-o (eval-last-sexp-and-go-lisp)*

$\overline{C-Z}$  *j (eval-dwim-lisp)*

$\overline{C-Z}$  *C-j (eval-dwim-and-go-lisp)*

Evaluate DWIM (Do What I Mean). If a region is selected, evaluate the region. If the cursor is on or immediately after a ')', evaluate the last sexp. If the cursor is on or immediately before a '(', evaluate the next sexp. If the cursor is inside a defun, evaluate the defun. If the cursor is inside a top-level sexp, evaluate the top-level sexp. Tests are done in the order specified in these comments, so if there is any ambiguity, make certain that the cursor is either on a parenthesis (for the eval last/next commands or not directly before/after/on a parenthesis for the eval defun/top-level commands).

$\overline{C-Z}$  *c (compile-defun-lisp)*

$\overline{C-Z}$  *C-c (compile-defun-lisp-and-go)*

When `compile-defun-lisp` is called in an inferior Lisp buffer with no current form, the last form typed to the top-level will be compiled.

$\overline{C-Z}$  *w (compile-region-lisp)*

$\overline{C-Z}$  *C-w (compile-region-and-go-lisp)*

If any of the forms contain an interactive command, then the command will never return. To get out of this state, you need to use `abort-commands-lisp` ( $\overline{C-Z}$  *g*). If `lisp-wait-p` is `t`, then EMACS will display the result of the command in the minibuffer or a pop-up window. If `lisp-wait-p` is `nil`, (the default) the send is done asynchronously and the results will be brought up only if there is more than one line or there is an error. In this case, you will be given the option of ignoring the error, keeping it in another buffer or keeping it and aborting all pending sends. If there is not a command already running in the inferior Lisp, you can preserve the break loop. If called with a negative prefix, the sense of `lisp-wait-p` will be inverted for the next command.

## 5.2 Documentation functions

`describe-lisp`, `inspect-lisp`, `arglist-lisp`, and `documentation-lisp` switch whether they prompt for a response or use a default when called with a negative prefix. If they

are prompting, there is completion through the inferior Lisp by using *TAB* or *M-TAB*. When entering an expression in the minibuffer, all of the normal ILISP commands like `arglist-lisp` also work.

Commands that work on a function will use the nearest previous function symbol. This is either a symbol after a ‘#’ or the symbol at the start of the current list.

The `fi:clman` and `fi:clman-apropos` commands for accessing the Franz Allegro CL documentation are not enabled by default. See Section 1.1 [Configuration and compilation], page 7.

**(C-z) a (arglist-lisp)**

Return the arglist of the current function. With a numeric prefix, the leading paren will be removed and the arglist will be inserted into the buffer.

**(SPC) (ilisp-arglist-message-lisp-space)**

Display the value of the argument list of a symbol followed by **(SPC)**. *To enable this feature you have to set `ilisp-arglist-message-lisp-space-p` to t.*

**(C-z) d (documentation-lisp)**

Infers whether function or variable documentation is desired. With a negative prefix, you can specify the type of documentation as well. With a positive prefix the documentation of the current function call is inserted into the buffer.

**(C-z) i (describe-lisp)**

Describe the previous sexp (it is evaluated). If there is no previous sexp and if called from inside an ILISP buffer, the previous result will be described.

**(C-z) I (inspect-lisp)**

Switch to the current inferior Lisp and inspect the previous sexp (it is evaluated). If there is no previous sexp and if called from inside an ILISP buffer, the previous result will be inspected.

**(C-z) H (hyperspec-lookup)**

Look up a standard symbol in the Common Lisp HyperSpec and display the relevant section.

**(C-z) L or (C-z) M-1 (cltl2-lookup)**

Look up a Common Lisp symbol in the CLtL2 book and display the relevant section.

**(C-z) D (fi:clman)**

**(C-z) A (fi:clman-*apropos*)**

If the Franz online Common Lisp manual is available, get information on a specific symbol. `fi:clman-apropos` will get information *apropos* a specific string. Some of the documentation is specific to the Allegro dialect, but most of it is for standard Common Lisp.

## 5.3 Macroexpansion

`(C-z) M (macroexpand-lisp)`

`(C-z) m (macroexpand-1-lisp)`

These commands apply to the next sexp. If called with a positive numeric prefix, the result of the macroexpansion will be inserted into the buffer. With a negative prefix, prompts for expression to expand.

## 5.4 Tracing functions

`(C-z) t (trace-defun-lisp)`

traces the current defun. When called with a numeric prefix the function will be untraced. When called with negative prefix, prompts for function to be traced.

`(C-z) C-t (trace-defun-lisp-break)`

traces the current defun and enters the debugger whenever that function is invoked. When called with a numeric prefix the function will be untraced. When called with negative prefix, prompts for function to be traced.

## 5.5 Package Commands

The first time an inferior Lisp mode command is executed in a Lisp Mode buffer, the package will be determined by using the regular expression `ilisp-hash-form-regexp` to find a package sexp and then passing that sexp to the inferior Lisp through `ilisp-package-command`. For the ‘common-lisp’ dialect, this will find the first `(in-package PACKAGE)` form in the file. A buffer’s package will be displayed in the mode line. If a buffer has no specification, forms will be evaluated in the current inferior Lisp package.

Buffer package caching can be turned off by setting the variable `lisp-dont-cache-package` to T. This will force ILISP to search for the closest previous "in-package" form corresponding to `ilisp-hash-form-regexp` in the buffer each time an inferior Lisp mode command is executed.

`(C-z) P (set-package-lisp)`

Set the inferior Lisp package to the current buffer’s package or with a prefix to a manually entered package.

`(C-z) p (set-buffer-package-lisp)`

Set the buffer’s package from the buffer. If it is called with a prefix, the package can be set manually.

## 5.6 Source Code Commands

The following commands all deal with finding things in source code. The first time that one of these commands is used, there may be some delay while the source module is loaded. When searching files, the first applicable rule is used:

- try the inferior Lisp,

- try a tags file if defined,
- try all buffers in one of `lisp-source-modes` or all files defined using `lisp-directory`.

`M-x lisp-directory` defines a set of files to be searched by the source code commands. It prompts for a directory and sets the source files to be those in the directory that match entries in `auto-mode-alist` for modes in `lisp-source-modes`. With a positive prefix, the files are appended. With a negative prefix, all current buffers that are in one of `lisp-source-modes` will be searched. This is also what happens by default. Using this command stops using a tags file.

`edit-definitions-lisp`, `who-calls-lisp`, and `edit-callers-lisp` will switch whether they prompt for a response or use a default when called with a negative prefix. If they are prompting, there is completion through the inferior Lisp by using `TAB` or `M-TAB`. When entering an expression in the minibuffer, all of the normal ILISP commands like `arglist-lisp` also work.

`edit-definitions-lisp` (`M-.`) will find a particular type of definition for a symbol. It tries to use the rules described above. The files to be searched are listed in the buffer `*Edit-Definitions*`. If `lisp-edit-files` is nil, no search will be done if not found through the inferior Lisp. The variable `ilisp-locator` contains a function that when given the name and type should be able to find the appropriate definition in the file. There is often a flag to cause your Lisp to record source files that you will need to set in the initialization file for your Lisp. The variable is `*record-source-files*` in both allegro and lucid. Once a definition has been found, `next-definition-lisp` (`M-,`) will find the next definition (or the previous definition with a prefix).

`edit-callers-lisp` (`(C-z) ^`) will generate a list of all of the callers of a function in the current inferior Lisp and edit the first caller using `edit-definitions-lisp`. Each successive call to `next-caller-lisp` (`M-'`) will edit the next caller (or the previous caller with a prefix). The list is stored in the buffer `*All-Callers*`. You can also look at the callers by doing `M-x who-calls-lisp`.

`search-lisp` (`M-?`) will search the current tags files, `lisp-directory` files or buffers in one of `lisp-source-modes` for a string or a regular expression when called with a prefix. `next-definition-lisp` (`M-,`) will find the next definition (or the previous definition with a prefix).

`replace-lisp` (`M-"`) will replace a string (or a regexp with a prefix) in the current tags files, `lisp-directory` files or buffers in one of `lisp-source-modes`.

Here is a summary of the above commands (behavior when given prefix argument is given in parentheses):

`M-x lisp-directory`

Define a set of files to be used by the source code commands.

`M-. (edit-definitions-lisp)`

Find definition of a symbol.

`M-, (next-definition-lisp)`

Find next (previous) definition.

`(C-z) ^ (edit-callers-lisp)`

Find all callers of a function, and edit the first.

M-‘ (next-caller-lisp)

Edit next (previous) caller of function set by `edit-callers-lisp`.

M-x who-calls-lisp

List all the callers of a function.

M-? (search-lisp)

Search for string (regular expression) in current tags, `lisp-directory` files or buffers. Use `next-definition-lisp` to find next occurrence.

M-" (replace-lisp)

Replace a string (regular expression) in files.

## 5.7 Batch commands

The following commands all deal with making a number of changes all at once. The first time one of these commands is used, there may be some delay as the module is loaded. The eval/compile versions of these commands are always executed asynchronously.

`mark-change-lisp` ( $\overline{C-Z}$  *SPC*) marks the current defun as being changed. A prefix causes it to be unmarked. `clear-changes-lisp` ( $\overline{C-Z}$  \* 0) will clear all of the changes. `list-changes-lisp` ( $\overline{C-Z}$  \* 1) will show the forms currently marked.

`eval-changes-lisp` ( $\overline{C-Z}$  \* e), or `compile-changes-lisp` ( $\overline{C-Z}$  \* c) will evaluate or compile these changes as appropriate. If called with a positive prefix, the changes will be kept. If there is an error, the process will stop and show the error and all remaining changes will remain in the list. All of the results will be kept in the buffer `*Last-Changes*`.

Summary:

$\overline{C-Z}$  *SPC* (`mark-change-lisp`)

Mark (unmark) current defun as changed.

$\overline{C-Z}$  \* e (`eval-changes-lisp`)

$\overline{C-Z}$  \* c (`compile-changes-lisp`)

Call with a positive prefix to keep changes.

$\overline{C-Z}$  \* 0 (`clear-changes-lisp`)

$\overline{C-Z}$  \* 1 (`list-changes-lisp`)

## 5.8 Files and directories

File commands in Lisp source-mode buffers keep track of the last used directory and file. If the point is on a string, that will be the default if the file exists. If the buffer is one of `lisp-source-modes`, the buffer file will be the default. Otherwise, the last file used in a `lisp-source-mode` will be used.

C-x C-f (`find-file-lisp`)

will find a file. If it is in a string, that will be used as the default if it matches an existing file. Symbolic links are expanded so that different references to the same file will end up with the same buffer.



**(C-z) l (load-file-lisp)**

will load a file into the inferior Lisp. You will be given the opportunity to save the buffer if it has changed and to compile the file if the compiled version is older than the current version. For ‘<whatever>.system’ files, which are used by DEFSYSTEM tools, no compilation or loading of possibly existing ‘<whatever>.binary-extension’ is attempted.

**(C-z) k (compile-file-lisp)**

will compile a file in the current inferior Lisp.

**(C-z) ! (default-directory-lisp)**

sets the default inferior Lisp directory to the directory of the current buffer. If called in an inferior Lisp buffer, it sets the Emacs `default-directory` to the Lisp default directory.

## 5.9 Switching between interactive and raw keyboard modes

There are two keyboard modes for interacting with the inferior Lisp, “interactive” and “raw”. Normally you are in interactive mode where keys are interpreted as commands to EMACS and nothing is sent to the inferior Lisp unless a specific command does so. In raw mode, all characters are passed directly to the inferior Lisp without any interpretation as EMACS commands. Keys will not be echoed unless `ilisp-raw-echo` is T.

Raw mode can be turned on interactively by the command `raw-keys-ilisp` (**(C-z) #**) and will continue until you type **(C-g)**. Raw mode can also be turned on/off by inferior Lisp functions if the command `io-bridge-ilisp` (`M-x io-bridge-ilisp`) has been executed in the inferior Lisp either interactively or on a hook. To turn on raw mode, a function should print `^[1^]` and to turn it off should print `^[0^]`. An example in Common Lisp would be:

```
(progn (format t "^[1^]") (print (read-char)) (format t "^[0^]"))
```

## 5.10 Interrupts, aborts, and errors

If you want to abort the last command you can use **C-g**.

If you want to abort all commands, you should use the command `abort-commands-lisp` (**(C-z) g**). Commands that are aborted will be put in the buffer `*Aborted Commands*` so that you can see what was aborted. If you want to abort the currently running top-level command, use `interrupt-subjob-ilisp` (**C-c C-c**). As a last resort, `M-x panic-lisp` will reset the ILISP state without affecting the inferior Lisp so that you can see what is happening.

`delete-char-or-pop-ilisp` (**C-d**) will delete prefix characters unless you are at the end of an ILISP buffer in which case it will pop one level in the break loop.

`reset-ilisp`, (**(C-z) z**) will reset the current inferior Lisp’s top-level so that it will no longer be in a break loop.

Summary:

**C-c C-c (interrupt-subjob-ilisp)**

Send a keyboard interrupt signal to lisp.

**$\overline{C-z}$  *g (abort-commands-lisp)***  
 Abort all running or unsent commands.

***M-x panic-lisp (panic-lisp)***  
 Reset the ILISP process state.

**$\overline{C-z}$  *z (reset-ilisp)***  
 Reset Lisp to top-level.

***C-d (delete-char-or-pop-ilisp)***  
 If at end of buffer, pop a level in break loop.

If `lisp-wait-p` is `nil` (the default), all sends are done asynchronously and the results will be brought up only if there is more than one line or there is an error. In case, you will be given the option of ignoring the error, keeping it in another buffer or keeping it and aborting all pending sends. If there is not a command already running in the inferior Lisp, you can preserve the break loop. If called with a negative prefix, the sense of `lisp-wait-p` will be inverted for the next command.

## 5.11 Interface to Lisp debuggers

ILD is an interface to Lisp debuggers, currently the ones of the AKCL, Allegro, CLISP, CMU CL, Corman Lisp and Lucid Common Lisp dialects. It uses a standard set of single-keystroke commands to interface to a variety of different debuggers and is vaguely modelled after the Symbolics debugger. It provides two key advantages: single keystrokes for moving up and down the stack, and a uniform interface to different debuggers.

Not all debugger commands are available in all implementations. Some are, but further work is needed. These are noted in the code (see the dialect definition files). If you know how to fix them please contact the ILISP maintainer.

Here is a list of the available ILD commands:

***M-a (ild-abort)***  
 Abort.

***M-c (ild-continue)***  
 Continue.



***M-C-n (ild-next)***  
 Next stack frame (with numeric argument *n*, move to the next *n*-th stack frame).

***M-C-p (ild-previous)***  
 Previous stack frame (with numeric argument *n*, move to the previous *n*-th stack frame).

***C-c < (ild-top)***  
 Top stack frame.

***C-c > (ild-bottom)***  
 Bottom stack frame.

***M-b (ild-backtrace)***  
 Backtrace.

- M-C-d (ild-locals)*  
Display all local variables.
- M-C-l (ild-local)*  
Display a particular local variable (with numeric argument *n*, display the *n*-th local variable).
- M-C-s (ild-step)*  
Step to the next breakpoint.
- C-c r (ild-return)*  
Return.
- M-C-r (ild-retry)*  
Retry.
- C-x t (ild-trap-on-exit)*  
Trap on exit.
- C-c L (select-lisp)*  
Select Lisp interaction buffer.
-  *C-s (slow-lisp)*  
Set compiler options for maximal debuggability.
-  *C-f (fast-lisp)*  
Set compiler options for fastest but least debuggable code.

## 5.12 Command history

ILISP mode is built on top of `comint-mode`, the general command-interpret-in-a-buffer mode. As such, it inherits many commands and features from this, including a command history mechanism.

Each ILISP buffer has a command history associated with it. Commands that do not match `ilisp-filter-regexp` and that are longer than `ilisp-filter-length` and that do not match the immediately prior command will be added to this history.

- M-n (comint-next-input)*  
*M-p (comint-previous-input)*  
Cycle through the input history.
- M-s (comint-previous-similar-input)*  
Cycle through input that has the string typed so far as a prefix.
- M-N (comint-psearch-input)*  
Search forwards for prompt.
- M-P (comint-msearch-input)*  
Search backwards for prompt.
- C-c R (comint-msearch-input-matching)*  
Search backwards for occurrence of prompt followed by string which is prompted for (*not* a regular expression).

See `comint-mode` documentation for more information on ‘`comint`’ commands.

## 5.13 Completion

Commands to reduce number of keystrokes.

### *M-TAB (complete-lisp)*

will try to complete the previous symbol in the current inferior Lisp. Partial completion is supported unless `ilisp-*prefix-match*` is set to `t`. (If you set it to `t`, inferior Lisp completions will be faster.) With partial completion, ‘p--n’ would complete to ‘position-if-not’ in Common Lisp. If the symbol follows a left paren or a ‘#’, only symbols with function cells will be considered. If the symbol starts with a ‘\*’ or you call with a positive prefix all possible completions will be considered. Only external symbols are considered if there is a package qualification with only one colon. The first time you try to complete a string the longest common substring will be inserted and the cursor will be left on the point of ambiguity. If you try to complete again, you can see the possible completions. If you are in a string, then filename completion will be done instead. And if you try to complete a filename twice, you will see a list of possible completions. Filename components are completed individually, so ‘/u/mi/’ could expand to ‘/usr/misc/’. If you complete with a negative prefix, the most recent completion (symbol or filename) will be undone.

### *M-RET (complete)*

will complete the current symbol to the most recently seen symbol in Emacs that matches what you have typed so far. Executing it repeatedly will cycle through potential matches. This is from the TMC completion package and there may be some delay as it is initially loaded.

## 5.14 Miscellany

Indentation, parenthesis balancing, movement and comment commands.

### *TAB (indent-line-ilisp)*

indents for Lisp. With prefix, shifts rest of expression rigidly with the current line.

### *M-C-q (indent-sexp-ilisp)*

will indent each line in the next sexp.

### *M-q (reindent-lisp)*

will reindent the current paragraph if in a comment or string. Otherwise it will close the containing defun and reindent it.

### *C-a (bol-ilisp)*

will go after the prompt as defined by `comint-prompt-regexp` or `ilisp-other-prompt` or to the left margin with a prefix.

### *DEL (backward-delete-char-untabify)*

converts tabs to spaces as it moves back.

### ; (comment-region-lisp)

will put prefix copies of `comment-start` before and `comment-end`’s after the lines in region. To uncomment a region, use a minus prefix.

C-z ) (*find-unbalanced-lisp*)

will find unbalanced parens in the current buffer. When called with a prefix it will look in the current region.



## 6 ILISP Customization

Starting a dialect runs the hooks on `comint-mode-hook` and `ilisp-mode-hook` and then *DIALECT-hooks* specific to dialects in the nesting order below.

```
common-lisp
  allegro
  clisp-hs
  cmulisp
  cormanlisp
  kcl
    akcl
      gcl
      ecl
    ibcl
  lispworks
  lucid
    liquid
  openmcl
  sbcl
scheme
  chez
  guile
  mzscheme
    drscheme-jr
  oaklisp
  Scheme->C (still "in fieri")
  scm
  snow
  stk
xlisp
  xlispsstat
```

On the very first prompt in the inferior Lisp, the hooks on `ilisp-init-hook` are run. For more information on creating a new dialect or variables to set in hooks, see ‘`ilisp.el`’.

ILISP Mode Hooks:

`ilisp-site-hook`

Executed when file is loaded

`ilisp-load-hook`

Executed when file is loaded

`ilisp-mode-hook`

Executed when an ilisp buffer is created

`ilisp-init-hook`

Executed after inferior Lisp is initialized and the first prompt is seen.

*DIALECT-hook*

Executed when dialect is set

Variables you might want to set in a hook or dialect:

```
ilisp-*prefix*
    Keys to prefix ilisp key bindings

ilisp-program
    Program to start for inferior Lisp

ilisp-motd
    String printed on startup with version

lisp-wait-p
    Set to t for synchronous sends

ilisp-handle-errors
    Set to t for ilisp to handle errors from the underlying Lisp.

ilisp-display-output-function
    The name of a function which displays ILISP output.

lisp-no-popper
    Set to t to have all output in inferior Lisp

ilisp-*use-frame-for-output*
    Set to t (default) to have multiline output in a distinct emacs-frame.

ilisp-*use-frame-for-arglist-output-p*
    Set to t (default) to have multiline arglist-output in a seperate emacs-frame.

ilisp-bindings-*bind-space-p*
    Set to t to have the SPC-key bound to #'ilisp-arglist-message-lisp-space.

ilisp-*arglist-message-lisp-space-p*
    Set to t to display the arglist of the current function displayed, after you hit
    SPC.

ilisp-*enable-imenu-p*
    Set to t to enable ilisp-imenu, that provides an index of all lisp-
    functions/definitions in a file.

lisp-show-status
    Set to nil to stop showing process status

ilisp-*prefix-match*
    Set to t if you do not want partial completion

ilisp-filter-regexp
    Input history filter

ilisp-filter-length
    Input history minimum length

ilisp-other-prompt
    Prompt for non- top-level read-eval print loops
```



## 7 Dialects

A *dialect* of Lisp is a specific implementation. For the parts of Common Lisp which are well specified, they are usually the same. For the parts that are not (debugger, top-level loop, etc.), there is usually the same functionality but different commands.

ILISP provides the means to specify these differences so that the ILISP commands will use the specific command peculiar to an implementation, but still offer the same behavior with the same interface.

### 7.1 Defining new dialects

To define a new dialect use the macro `defdialect`. For examples, look at the dialect definitions in `'ilisp-acl.el'`, `'ilisp-cmu.el'`, `'ilisp-kcl.el'`, `'ilisp-luc.el'`. There are hooks and variables for almost anything that you are likely to need to change. The relationship between dialects is hierarchical with the root values being defined in `setup-ilisp`. For a new dialect, you only need to change the variables that are different than in the parent dialect.

### 7.2 Writing new commands

Basic tools for creating new commands:

`deflocal` Define a new buffer local variable.

`ilisp-dialect`  
List of dialect types. For specific dialect clauses.

`lisp-symbol`  
Create a symbol.

`lisp-symbol-name`  
Return a symbol's name

`lisp-symbol-delimiter`  
Return a symbol's qualification

`lisp-symbol-package`  
Return a symbol's package

`lisp-string-to-symbol`  
Convert string to symbol

`lisp-symbol-to-string`  
Convert symbol to string

`lisp-buffer-symbol`  
Convert symbol to string qualified for buffer

`lisp-previous-symbol`  
Return previous symbol

`lisp-previous-sexp`

Return previous sexp

`lisp-def-name`

Return name of current definition

`lisp-function-name`

Return previous function symbol

`ilisp-read`

Read an sexp with completion, arglist, etc

`ilisp-read-symbol`

Read a symbol or list with completion

`ilisp-completing-read`

Read from choices or list with completion

Notes:

- Special commands like `arglist` should use `ilisp-send` to send a message to the inferior Lisp.
- Eval/compile commands should use `eval-region-lisp` or `compile-region-lisp`.

# Concept Index

## \*

*Aborted Commands* buffer	15, 23
*All-Callers* buffer	15, 21
*Arglist-Output* buffer	15
*Changed-Definitions* buffer	15
*Completions* buffer	15
*Edit-Definitions* buffer	15, 21
*Error Output* buffer	15
*Errors* buffer	15
*ilisp-send* buffer	15
*Last-Changes* buffer	15, 22
*Output* buffer	15

## .

‘.emacs’	7
‘.lisp’ files	8

## A

Aborting commands	23
Aborting from a debugger	24
Allegro CL	9, 11, 19
‘and-go’ functions	17
Anonymous FTP	1
ANSI Common Lisp	8
Apropos help	19
Arglist Lisp	19
Austin Kyoto Common Lisp	11

## B

Backtrace	24
Bottom stack frame	24
Break loop	23
browse-url	8, 9
Buffer package	20
Buffer package caching	20
buffers of ILISP	15
Bugs, reporting them	1
bury output window	16
Byte-compiling ILISP files	7

## C

Call	17
Change commands	22
Chez Scheme	11
Clearing changes	22
CLISP	11
CLtL2	8, 19
CMU Common Lisp	11
comint-mode	25
Command history	25

Comment region	26
Common Lisp	8, 11
Common Lisp HyperSpec	8, 19
Common Lisp manual	19
Common Lisp Manual	8
Compile last form	18
Compile region	18
Compile/eval commands	17
Compiler options	25
Compiling changes	22
Compiling files	23
Compiling ILISP files	7
Completion	26
Continuing from a debugger	24
Converting tabs to spaces	26
‘COPYING’	1
Corman Lisp	11
Current directory	22
Currently running command	17
Customization	29

## D

Debugger interface	24
Default directory	23
‘default.el’	7
Defining new dialects	31
DEFSYSTEM files	23
Defun	17
Describing bindings	17
Describing Lisp objects	19
Dialect startup	29
Dialects	31
Dialects supported	11
Directories and files	22
Displaying commands	17
Displaying local variables	25
‘docs’	9
Documentation	9
Documentation Functions	19
Downloading ILISP	1
DrScheme-jr	11

## E

easymenu package	7
EcoLisp	11
Errors	23
Eval region	18
Eval’ing changes	22
Eval/compile commands	17
Expanding macro forms	20
‘extra’	8

**F**

features .....	5
File changes .....	22
Filename completion .....	26
Files and directories .....	22
Find callers .....	21
Find file .....	22
Find unbalanced parens .....	27
Finding source .....	20
First prompt .....	29
Franz manual .....	9, 19
FSF keyspace .....	13
FTP site .....	1

**G**

Getting ILISP .....	1
GNU Common Lisp .....	11
Going after the prompt .....	26
Group changes .....	22
grow output window .....	16
GUILE .....	11

**H**

Hooks .....	29
How to get .....	1
HyperSpec .....	8, 19

**I**

Ibuki Common Lisp .....	11
'icompile.bat' .....	7
ILD .....	24
ILISP buffers .....	15
Ilisp menu .....	7
ILISP Mode Hooks .....	29
'ilisp-all.elc' .....	7
ilisp-announce mailing list .....	1
ilisp-cvs mailing list .....	1
'ilisp-def.el' .....	7, 9
ilisp-devel mailing list .....	1, 3
ilisp-help mailing list .....	1, 2
'ilisp-s2c.el' .....	12
'ilisp.emacs' .....	7, 8, 9
In-package form .....	20
Indentation .....	26
Input search .....	25
Inserting calls .....	17
Inserting results .....	17
Installation .....	7
'INSTALLATION' .....	7
Interactive keyboard mode .....	23
Interface to Lisp debuggers .....	24
Internal ILISP functions .....	31
Interrupting commands .....	23

**K**

keybindings .....	13
Kyoto Common Lisp .....	11

**L**

Last command .....	25
Lisp find file .....	22
Lisp menu .....	7
List callers .....	21
Listing bindings .....	17
Listing changes .....	22
Loading files .....	23
Local variables .....	25
Lucid Common Lisp .....	11

**M**

Macroexpansion .....	20
Mailing lists .....	1
'Makefile' .....	7, 9
Marking changes .....	22
Minibuffer completion .....	19
Modeline status .....	17
MzScheme .....	11

**N**

Negative prefix .....	19
Next definition .....	21
Next input .....	25
Next stack frame .....	24

**O**

Oaklisp .....	11
OpenMCL .....	11

**P**

Package commands .....	20
Parenthesis balancing .....	27
Partial completion .....	26
Pop in break loop .....	23
Previous commands .....	25
Previous definition .....	21
Previous lisp buffer .....	16
Previous stack frame .....	24

**R**

Raw keyboard mode .....	23
Reducing Emacs startup time .....	7
Reference card .....	9
Region commands .....	17
Reindent lisp .....	26
Replace lisp .....	21
Reporting bugs .....	1
Resetting Lisp .....	23
Retrying from a debugger .....	25
Returning from a debugger .....	25
Rigid indentation .....	26
Running Lisp .....	11

**S**

SB Common Lisp .....	11
Scheme .....	11
Scheme->C .....	12
SCM .....	11
scrolling output .....	16
Search input .....	25
Selecting a Lisp interaction buffer .....	25
Sending input to Lisp .....	17
Set buffer package .....	20
Set default directory .....	23
Setting compiler options .....	25
Similar input .....	25
Snow .....	11
Source Code Commands .....	20
Source modes .....	21
Stack backtrace .....	24
Stack frames .....	24
Starting up Lisp .....	11

Startup time, how to reduce it .....	7
Status light .....	17
Stepping to next breakpoint .....	25
STk .....	11
Supported dialects .....	11
Switching buffers .....	16
Symbolic link expansion .....	22
System definition files .....	23

**T**

TMC completion .....	26
Top stack frame .....	24
Top-level, return to .....	23
Tracing defuns .....	20
Trapping on exit from a debugger .....	25
Turning off typeout windows .....	16
Typeout windows .....	15

**U**

Uncomment region .....	26
Untracing defuns .....	20
User manual .....	9

**W**

Web site .....	1
Windows, compiling under .....	7

**X**

XLisp .....	11, 12
XLisp-Stat .....	11, 12



# Key Index

## C

C-]	17
C-a	26
C-c <	24
C-c >	24
C-c L	25
C-c r	25
C-c R	25
C-d	23
C-g	23
C-x C-f	22
C-x t	25
!	23
#	23
)	27
* 0	22
* c	22
* e	22
* l	22
;	26
^	21
1	16
a	19
A	19
b	16
c	18
C-c	18
C-e	18
C-f	25
C-j	18
C-n	18
C-o	18
C-r	18
C-s	25
C-t	20
C-w	18
d	19
D	19
e	18
g	18, 23
G	16
H	19
i	19
I	19
j	18
k	23
l	23
L	19
m	20
M	20
M-l	19
n	18
o	18
p	20
P	20
prefix	17
r	18

s	17
SPC	22
t	20
v	16
w	18
z	23

## D

DEL	26
-----	----

## L

LFD	18
-----	----

## M

M-"	21
M-,	21
M-	21
M-?	21
M-'	21
M-a	24
M-b	24
M-c	24
M-C-d	25
M-C-l	16, 25
M-C-n	24
M-C-p	24
M-C-q	26
M-C-r	25
M-C-s	25
M-C-x	18
M-n	25
M-N	25
M-p	25
M-P	25
M-q	26
M-RET	26
M-s	25
M-TAB	19, 26
M-x io-bridge-ilisp	23
M-x lisp-directory	21
M-x who-calls-lisp	21

## R

RET	17
-----	----

## S

	19
--	----

## T

TAB	19
TAB	26





# Command Index

Commands available via *M-x* prefix.

## A

abort-commands-lisp ..... 18, 23  
 akcl ..... 11  
 allegro ..... 11  
 arglist-lisp ..... 19

## B

backward-delete-char-untabify ..... 26  
 bol-ilisp ..... 26

## C

chez ..... 11  
 clear-changes-lisp ..... 22  
 clisp-hs ..... 11  
 close-and-send-lisp ..... 17  
 cltl2-lookup ..... 19  
 cmulisp ..... 11  
 comint-msearch-input ..... 25  
 comint-msearch-input-matching ..... 25  
 comint-next-input ..... 25  
 comint-previous-input ..... 25  
 comint-previous-similar-input ..... 25  
 comint-psearch-input ..... 25  
 comment-region-lisp ..... 26  
 common-lisp ..... 11  
 compile-changes-lisp ..... 22  
 compile-defun-and-go-lisp ..... 17  
 compile-defun-lisp ..... 18  
 compile-defun-lisp-and-go ..... 18  
 compile-file-lisp ..... 23  
 compile-region-and-go-lisp ..... 18  
 compile-region-lisp ..... 18  
 complete ..... 26  
 complete-lisp ..... 26  
 cormanlisp ..... 11

## D

default-directory-lisp ..... 23  
 defdialect ..... 31  
 delete-char-or-pop-ilisp ..... 23  
 describe-lisp ..... 19  
 documentation-lisp ..... 19  
 drscheme-jr ..... 11

## E

eccl ..... 11  
 edit-callers-lisp ..... 21  
 edit-definitions-lisp ..... 21  
 eval-changes-lisp ..... 22  
 eval-defun-and-go-lisp ..... 17, 18  
 eval-defun-lisp ..... 18  
 eval-dwim-and-go-lisp ..... 18  
 eval-dwim-lisp ..... 18  
 eval-last-sexp-and-go-lisp ..... 18  
 eval-last-sexp-lisp ..... 18  
 eval-next-sexp-and-go-lisp ..... 18  
 eval-next-sexp-lisp ..... 18  
 eval-region-and-go-lisp ..... 18  
 eval-region-lisp ..... 18

## F

fast-lisp ..... 25  
 fi:clman ..... 9, 19  
 fi:clman-apropos ..... 19  
 find-file-lisp ..... 22  
 find-unbalanced-lisp ..... 27

## G

gcl ..... 11  
 guile ..... 11

## H

hyperspec-lookup ..... 19

## I

ibcl ..... 11  
 ild-abort ..... 24  
 ild-backtrace ..... 24  
 ild-bottom ..... 24  
 ild-continue ..... 24  
 ild-local ..... 25  
 ild-locals ..... 25  
 ild-next ..... 24  
 ild-previous ..... 24  
 ild-retry ..... 25  
 ild-return ..... 25  
 ild-step ..... 25  
 ild-top ..... 24  
 ild-trap-on-exit ..... 25  
 ilisp-arglist-message-lisp-space ..... 19  
 ilisp-bug ..... 1  
 ilisp-bury-output ..... 16

ilisp-compile-inits.....	8
ilisp-grow-output.....	16
ilisp-scroll-output.....	16
indent-line-ilisp.....	26
indent-sexp-ilisp.....	26
inspect-lisp.....	19
interrupt-subjob-ilisp.....	23
io-bridge-ilisp.....	23

## K

kcl.....	11
----------	----

## L

lisp-directory.....	21
list-changes-lisp.....	22
load-file-lisp.....	23
lucid.....	11

## M

macroexpand-1-lisp.....	20
macroexpand-lisp.....	20
make.....	7, 9
make compile.....	7
make docs.....	9
make dvi.....	9
make html.....	9
make info.....	9
make loadfile.....	7
make ps.....	9
mark-change-lisp.....	22
mzscheme.....	11

## N

newline-and-indent-lisp.....	18
next-caller-lisp.....	21
next-definition-lisp.....	21

## O

oaklisp.....	11
openmcl.....	11

## P

panic-lisp.....	23
previous-buffer-lisp.....	16

## R

raw-keys-ilisp.....	23
reindent-lisp.....	26
replace-lisp.....	21
reset-ilisp.....	23
return-ilisp.....	17
run-ilisp.....	11

## S

sbcl.....	11
scheme.....	11
scm.....	11
search-lisp.....	21
select-lisp.....	25
set-buffer-package-lisp.....	20
set-package-lisp.....	20
setup-ilisp.....	31
slow-lisp.....	25
snow.....	11
status-lisp.....	17
stk.....	11
switch-to-lisp.....	16

## T

trace-defun-lisp.....	20
trace-defun-lisp-break.....	20

## W

who-calls-lisp.....	21
---------------------	----

## X

xlisp.....	11
xlispstat.....	11

# Variable Index

Variables and hooks of ILISP.

## \*

*\*record-source-files\** ..... 21

## A

*auto-mode-alist* ..... 21

## C

*comint-always-scroll* ..... 16

*comint-mode-hook* ..... 29

*comint-prompt-regexp* ..... 26

## D

*default-directory* ..... 23

*DIALECT-hook* ..... 29

## E

*EMACS* ..... 7

## H

*HyperSpec* ..... 7

## I

*ilisp-arglist-message-lisp-space-p*... 19, 30

*ilisp-enable-cl-easy-menu-p* ..... 7

*ilisp-enable-imenu-p* ..... 30

*ilisp-enable-scheme-easy-menu-p* ..... 7

*ilisp-prefix\** ..... 17, 30

*ilisp-prefix-match\** ..... 26, 30

*ilisp-use-fi-clman-interface-p* ..... 9

*ilisp-use-frame-for-arglist-output-p*... 30

*ilisp-use-frame-for-output\** ..... 30

*ilisp-use-fsf-compliant-keybindings\** .... 13

*ilisp-bindings-bind-space-p* ..... 30

*ilisp-defvar-regexp* ..... 17

*ilisp-display-output-function* ..... 30

*ilisp-filter-length* ..... 25, 30

*ilisp-filter-regexp* ..... 25, 30

*ilisp-handle-errors* ..... 30

*ilisp-hash-form-regexp* ..... 20

*ilisp-init-binary-command* ..... 7

*ilisp-init-binary-extension* ..... 7

*ilisp-init-hook* ..... 29

*ilisp-load-hook* ..... 29

*ilisp-load-inits* ..... 7

*ilisp-locator* ..... 21

*ilisp-mode-hook* ..... 29

*ilisp-motd* ..... 30

*ilisp-other-prompt* ..... 26, 30

*ilisp-program* ..... 7, 30

*ilisp-raw-echo* ..... 23

*ilisp-site-hook* ..... 7, 8, 29

## L

*lisp-dont-cache-package* ..... 20

*lisp-edit-files* ..... 21

*lisp-no-popper* ..... 16, 30

*lisp-show-status* ..... 17, 30

*lisp-source-modes* ..... 21

*lisp-wait-p* ..... 18, 24, 30

*LN* ..... 7

## P

*pop-up-windows* ..... 16



# Function Index

Internal functions of ILISP which can be used to write new commands.

## C

compile-region-lisp ..... 32

## D

deflocal ..... 31

## E

eval-region-lisp ..... 32

## I

ilisp-compile-inits ..... 7

ilisp-completing-read ..... 32

ilisp-dialect ..... 31

ilisp-package-command ..... 20

ilisp-read ..... 32

ilisp-read-symbol ..... 32

ilisp-send ..... 32

## L

lisp-buffer-symbol ..... 31

lisp-def-name ..... 32

lisp-function-name ..... 32

lisp-previous-sexp ..... 32

lisp-previous-symbol ..... 31

lisp-string-to-symbol ..... 31

lisp-symbol ..... 31

lisp-symbol-delimiter ..... 31

lisp-symbol-name ..... 31

lisp-symbol-package ..... 31

lisp-symbol-to-string ..... 31



# Table of Contents

<b>How to get the latest ILISP distribution.....</b>	<b>1</b>
FTP and Web directions.....	1
<b>Acknowledgements.....</b>	<b>3</b>
<b>Features.....</b>	<b>5</b>
<b>1 How to install ILISP.....</b>	<b>7</b>
1.1 Configuration and compilation.....	7
<b>2 How to run a Lisp process using ILISP.....</b>	<b>11</b>
<b>3 A word about the keys used by ILISP.....</b>	<b>13</b>
<b>4 Buffers used by ILISP, and their commands</b>	
.....	<b>15</b>
4.1 Typeout windows.....	15
4.2 Switching buffers.....	16
<b>5 ILISP Commands.....</b>	<b>17</b>
5.1 Eval and compile functions.....	17
5.2 Documentation functions.....	18
5.3 Macroexpansion.....	19
5.4 Tracing functions.....	20
5.5 Package Commands.....	20
5.6 Source Code Commands.....	20
5.7 Batch commands.....	22
5.8 Files and directories.....	22
5.9 Switching between interactive and raw keyboard modes ...	23
5.10 Interrupts, aborts, and errors.....	23
5.11 Interface to Lisp debuggers.....	24
5.12 Command history.....	25
5.13 Completion.....	25
5.14 Miscellany.....	26
<b>6 ILISP Customization.....</b>	<b>29</b>
<b>7 Dialects.....</b>	<b>31</b>
7.1 Defining new dialects.....	31
7.2 Writing new commands.....	31

<b>Concept Index . . . . .</b>	<b>33</b>
<b>Key Index . . . . .</b>	<b>37</b>
<b>Command Index . . . . .</b>	<b>39</b>
<b>Variable Index . . . . .</b>	<b>41</b>
<b>Function Index . . . . .</b>	<b>43</b>