

pLisp User Manual

1. Introduction

pLisp is an integrated development environment for Lisp. While it aims to be a friendly Lisp development system for beginners, its feature-set is comprehensive enough to address the needs of a small-to-medium sized Lisp project. Please refer to the README file for more details. This document is the user manual for pLisp. It is not a language reference manual. Email me at rajesh.jayaprakash@gmail.com for comments and suggestions.

2. Getting pLisp

You can get pLisp binaries from [here](#). Alternately you can either use git to check out the code.

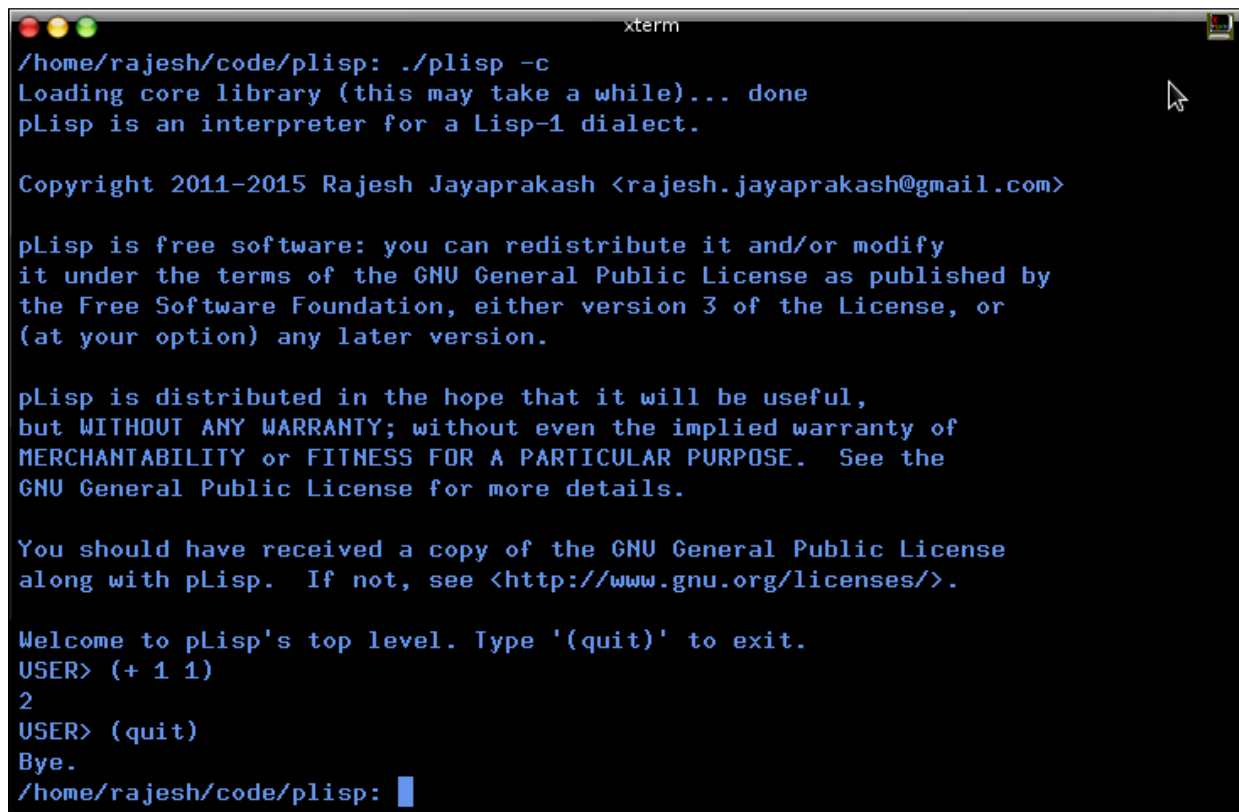
3. Installing pLisp

Please follow [these](#) instructions to install pLisp.

4. Invoking pLisp

You have four options while starting pLisp:

- a. **plisp -c**: Run pLisp in command-line mode, i.e., without any of the user interface bells and whistles. You will be presented with just a prompt, and you have to type your Lisp expressions at this prompt (Figure 1). The results of evaluating the expression will be displayed below the prompt (more later in Section 13 on what the prompt signifies). After you're done with all the expressions, type '(quit)' to exit pLisp. Then do a hundred push-ups and take a cold shower.
- b. **plisp -e <expression>**: Use pLisp to just evaluate a single expression, print the results and

A terminal window titled 'xterm' showing the execution of the pLisp interpreter. The user runs './plisp -c' at the prompt '/home/rajesh/code/plisp:'. The output includes a loading message, a copyright notice for Rajesh Jayaprakash (2011-2015), and the GNU General Public License text. The user then enters '(+ 1 1)' and '(quit)' at the 'USER>' prompt, resulting in the output '2' and 'Bye.' respectively. The prompt returns to '/home/rajesh/code/plisp:'.

```
/home/rajesh/code/plisp: ./plisp -c
Loading core library (this may take a while)... done
pLisp is an interpreter for a Lisp-1 dialect.

Copyright 2011-2015 Rajesh Jayaprakash <rajesh.jayaprakash@gmail.com>

pLisp is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

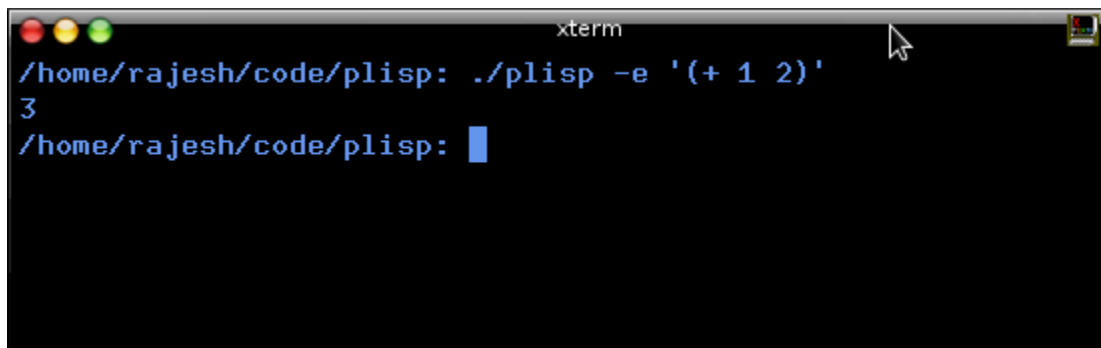
pLisp is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with pLisp. If not, see <http://www.gnu.org/licenses/>.

Welcome to pLisp's top level. Type '(quit)' to exit.
USER> (+ 1 1)
2
USER> (quit)
Bye.
/home/rajesh/code/plisp: 
```

Figure 1

quit (see Figure 2). This is useful for both running the programs that you have created using pLisp, and for using pLisp as a calculator [**plisp -e '(+ 1 2)'**]. Note that the expression has to be enclosed in single or double quotes.

A terminal window titled 'xterm' showing the execution of the pLisp interpreter with the -e flag. The user runs './plisp -e '(+ 1 2)'' at the prompt '/home/rajesh/code/plisp:'. The output is '3', and the prompt returns to '/home/rajesh/code/plisp:'.

```
/home/rajesh/code/plisp: ./plisp -e '(+ 1 2)'
3
/home/rajesh/code/plisp: 
```

Figure 2

c. **plisp -i <image file>**: Start pLisp by loading an existing image. An image is a file on your hard disk that stores a snapshot of your pLisp system – all the functions, variables, and other objects that you created in the course of your development. Image-based development is not only an incredibly powerful way to build code, but also has other benefits (easier debugging and persistent objects, to name a few). Also, the cool thing about pLisp images is that they remember the details of the UI elements, so if you open an image, you will be presented with the same windows (even the Debugger window) that were open when you saved the image last.

d. **pLisp -p**: Run pLisp in pipe mode. This is an experimental feature; this mode is similar to command-line mode, except that no prompts are printed. It is for scenarios where you might want to interface to pLisp with another executable.

These options can be used alone or in combination: you can start with an image in command line mode, or evaluate a single expression on an image and quit. But you cannot use both the '-c' (or '-p') and '-e' options together: if you're going to quit after evaluating a single expression, it doesn't matter whether you are in command-line mode or GUI mode.

The order of these options also does not matter; for example, you can either say **plisp -e <expression> -i <image file>** or **plisp -i <image file> -e <expression>**.

Note: If you invoke pLisp without any arguments, it looks for the default image file named 'plisp.image' in the current directory, and if this file is found, proceeds as if it was invoked with the '-i ./plisp.image' option. If plisp.image is not found, it starts up in 'no-image' mode (warning: this will take a while, since all the core library forms will have to be compiled from scratch); you will then have to save/create the image manually yourself when you exit pLisp. pLisp ships with this default image file as part of the distribution, copy this file to a location of your choice.

5. The Transcript Window

So you have invoked pLisp in GUI mode. You will be presented with the main window in the pLisp UI, namely the Transcript window (Figure 3).

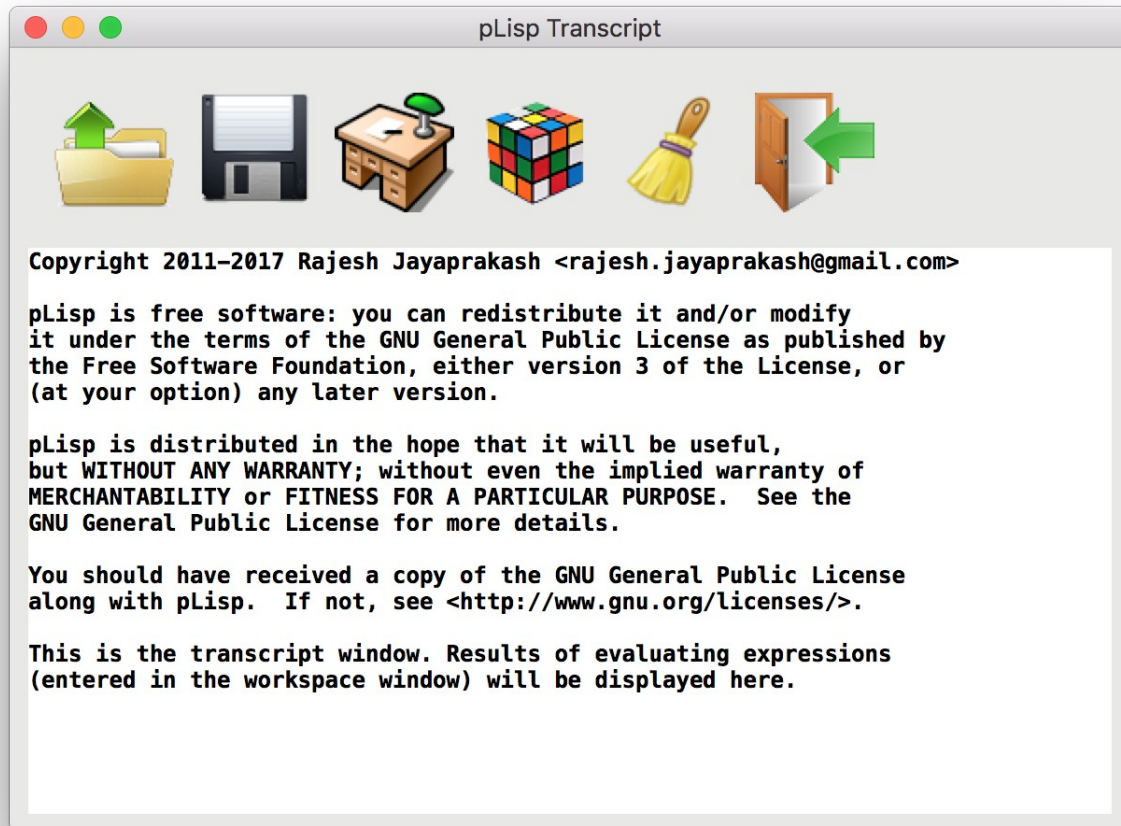


Figure 3

The Transcript window displays the results of evaluating expressions entered in the Workspace window. In addition to this, it's also the primary means of controlling pLisp, through the tool bar actions. The tool bar has six buttons:



Click on this button if pLisp has been invoked without an image file and you would like to load an image file, or if you would like to switch to another image file. Keyboard shortcut: Control-L



Click on this button either to save an already loaded image file or to create a fresh image. Keyboard shortcut: Control-S.



This button opens the Workspace window or, if it's already open, brings it to the front. Keyboard shortcut: F7.



This button opens the System Browser window or, if it's already open, brings it to the front. Keyboard shortcut: F9.



Click on this button to clear the contents of the Transcript.



Exits pLisp. If you have made changes to an image, you will be prompted to save the changes. Keyboard shortcut: Control-W.

6. The Workspace Window

As mentioned in the previous section, pressing F7 or clicking on the Workspace tool bar button in the Transcript window brings up the Workspace window (Figure 4).

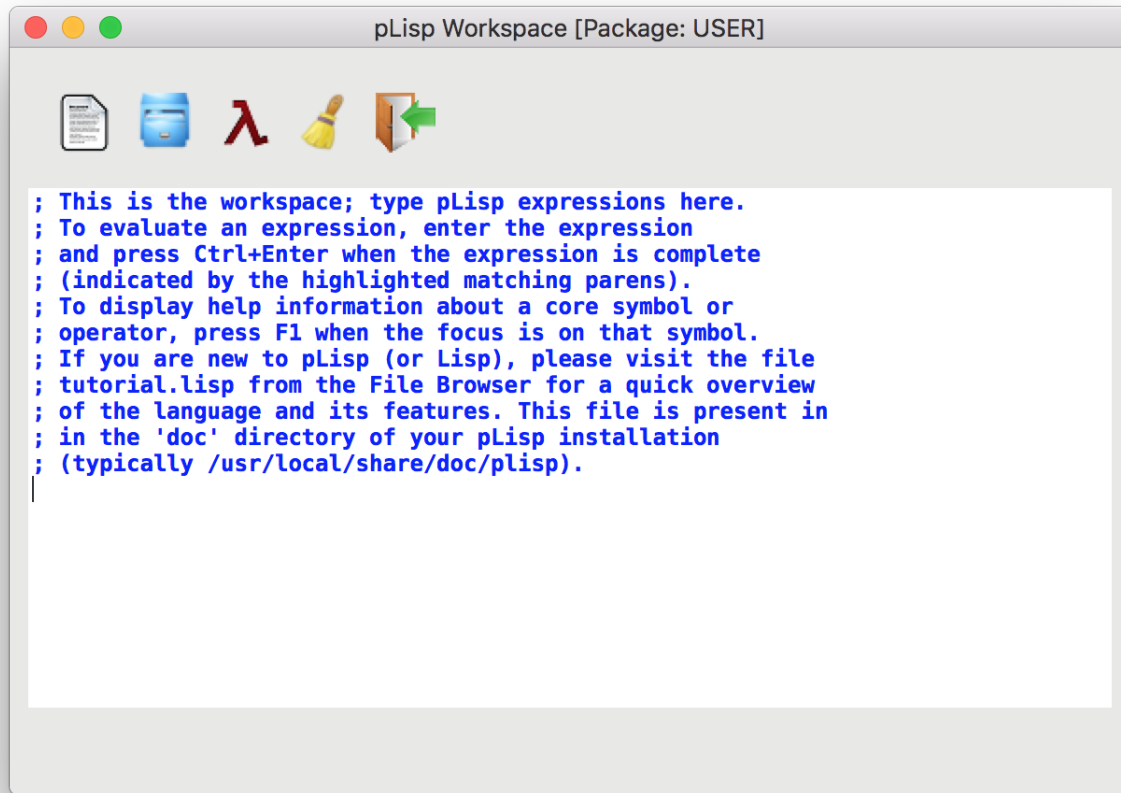


Figure 4

As the text in the window helpfully indicates, type the pLisp expressions here that you would like evaluated. The result will appear in the Transcript window if all goes well. The Workspace window has five tool bar buttons that can be used to control its behaviour:



Click on this button to load a file containing pLisp source expressions. Useful if you would like to feed multiple (previously saved) expressions in one shot. Keyboard shortcut: Control-O.



This button opens the File Browser window, or, if it's already opened, brings it to the front.



Click on this button to evaluate a selected expression entered into the workspace. Keyboard shortcut: Control-Enter.



Click on this button to clear the contents of the workspace.



Click on this button to close the Workspace Window. Keyboard shortcut: Control-W.

7. System Browser Window

Pressing F9 or clicking on the System Browser tool bar button in the Transcript window brings up the System Browser Window (Figure 5).

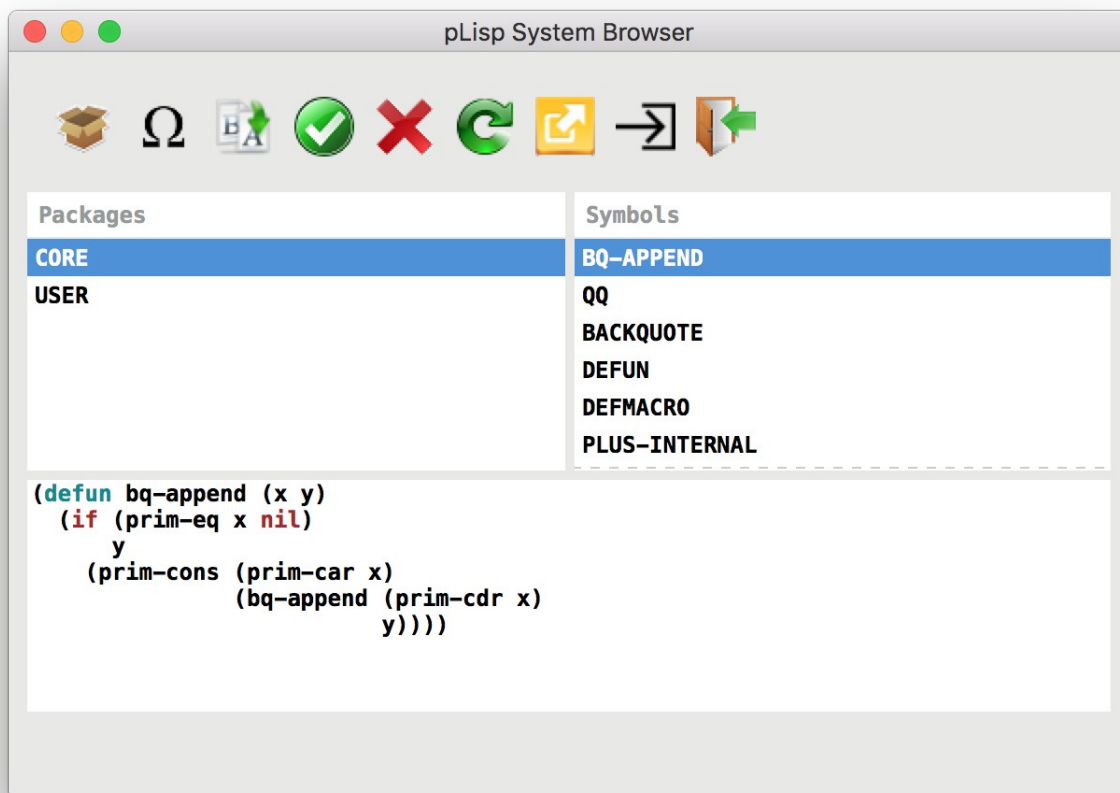


Figure 5

The System browser has three panels: the Packages panel, the Symbols panel, and the code panel. The Packages panel lists all the packages defined in the system (see Section 13). The Symbols panel lists all the symbols belonging to the selected package. Finally, the code panel displays the code corresponding to the selected symbol if it's a closure or a macro, or the value (integer, float, character, etc.) that is bound to the symbol.

The System Browser has these nine tool bar buttons:



Click on this button to create a new package. Keyboard shortcut: Control-K.



Click on this button to rename a symbol. Keyboard shortcut: F2.



Click on this button to create a new symbol in the selected package. Keyboard shortcut: Control-N.



Click on this button to accept the changes made in the code panel. Keyboard shortcut: Control-S.



Click on this button to delete the selected symbol. Keyboard shortcut: Control-X.



Click on this button to refresh the contents of the System Browser (for example, if you've made changes by typing expressions in the Workspace window and would like to sync the System Browser with these changes). Keyboard shortcut: F5.



Click on this button to export the contents (in pLisp source form) of the selected package



Click on this button to list all the top-level definitions that refer to the selected symbol



Click on the button to close the System Browser window. Keyboard shortcut: Control-W.

8. Referrers Window

The Referrers window is invoked from the System Browser window to list all the top level definitions that refer to the selected symbol. Figure 6 below shows a sample invocation of the Referrers window for the core symbol 'length':

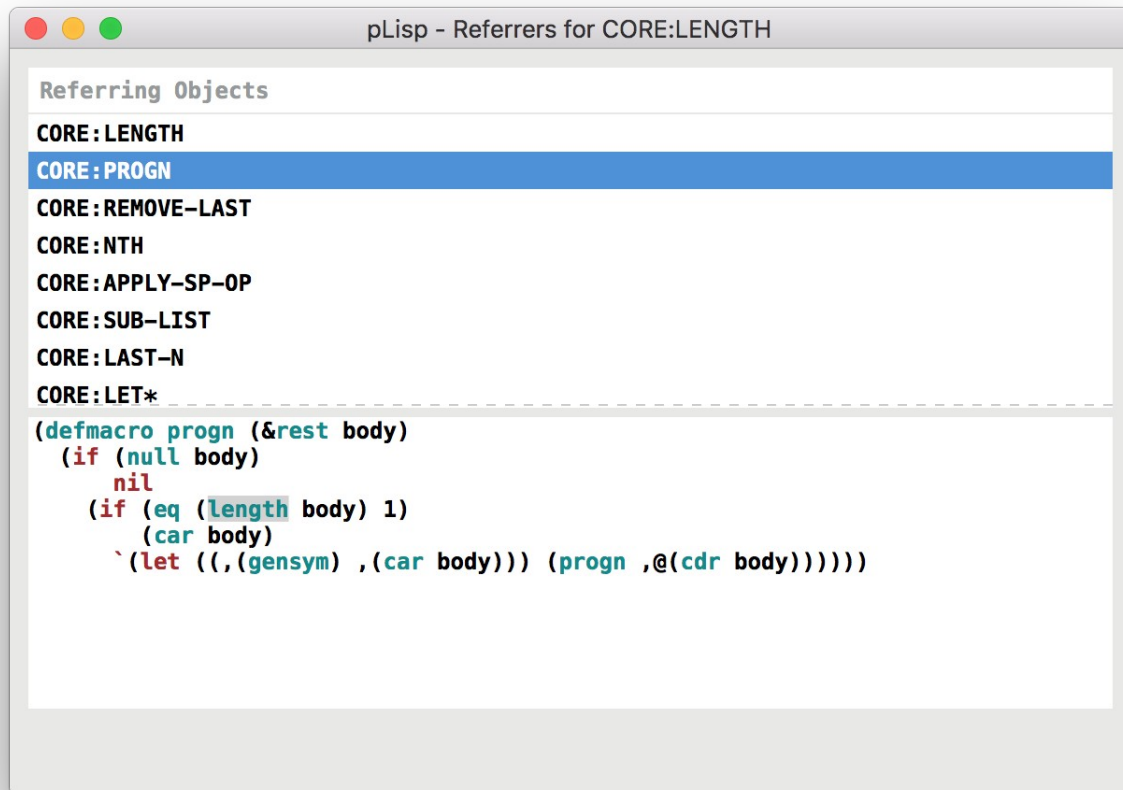


Figure 6

9. File Browser Window

The File Browser window is invoked from the Workspace window. You can view and edit pLisp source files using the File Browser. The File Browser also supports evaluating expressions by using either the keyboard shortcut (Control-Enter) or the provided toolbar button.

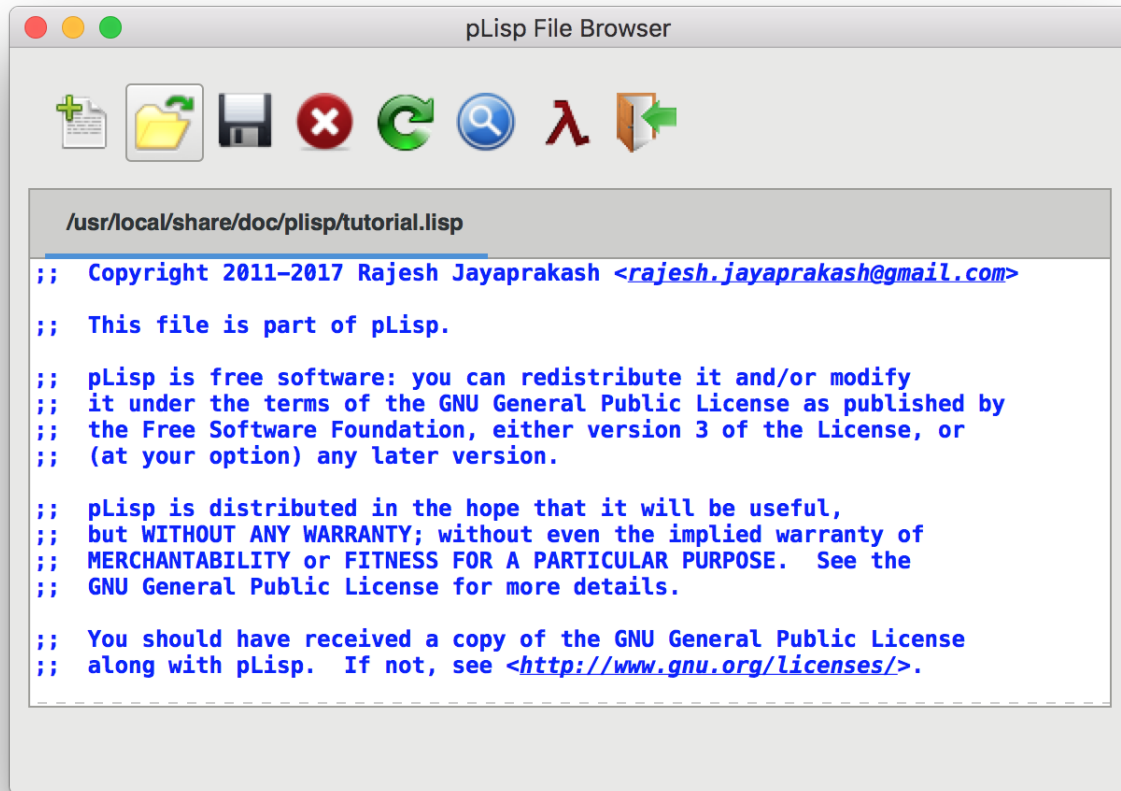


Figure 7

Note: Press Control-F to trigger the search window in the File Browser, press Control-G to continue the search.

10. Stepper Window

The Stepper window can be used to step into the execution of an expression, and examine the values of the arguments and local variables. It is invoked by using the `step` special operator. At any time during the stepping, you can choose to run the execution to completion, step over an expression, or abort the execution. Please note that if your code contains `break` statements, the stepper will still stop at these statements even if you opted to run the execution to completion. Figure 8 shows the Stepper in action, stepping through the expression

```
(let ((x 0))
  (print x)
  (break)
  (incf x)
  (break))
```

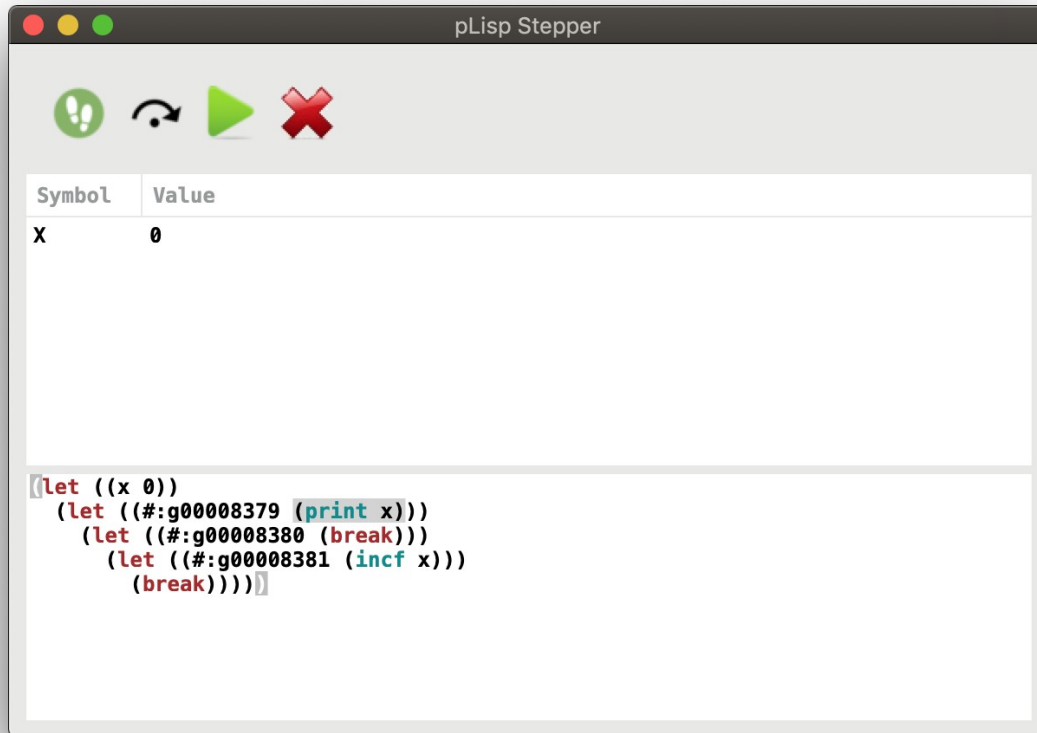


Figure 8

Since the body of the `let` expression is implicitly converted into a `progn` (which is a macro in pLisp), what you see in the Stepper window is somewhat different from the actual expression typed in.

11. Debugger Window

(Note: The Debugger feature has been superseded by the Stepper functionality (see previous section), and is retained only for providing more meaningful feedback when the code hits an uncaught exception.)

Debugging pLisp code entails placing '`BREAK`' statements judiciously in your expressions and inspecting the state of the system by means of the Debugger window. The Debugger window displays the call stack.

A (contrived) example will clarify things.

```
(let ((x 0))
  (print x)
  (break)
  (incf x)
  (break))
```

This code creates a local variable `x`, initialized to zero, prints it, then increments it. There are break statements after the print and the increment. Evaluating this expression brings up the Debugger window twice (Figure 7), once for each break statement (pressing the Resume button in the window resumes the evaluation). Pressing the Abort button aborts the evaluation.

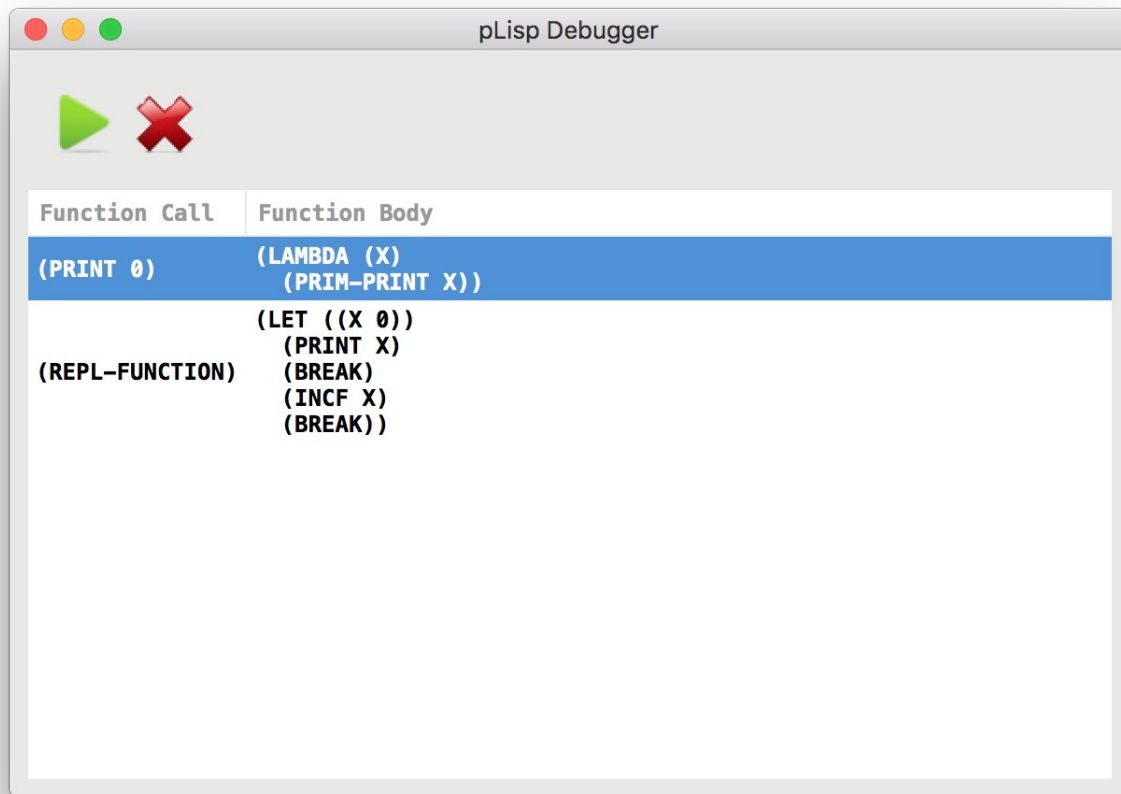


Figure 9

12. Exceptions

The Debugger window is also invoked automatically whenever your code hits an exception. The call stack is displayed, with the expression causing the exception on top (Figures 10, 11 and 12). An important point to note is that since the pLisp code is compiled into native code using continuation-passing style, no function ever returns, and the call stack includes all the function objects invoked directly and indirectly through the REPL invocation. This can be somewhat unnerving; it is advised that you use the Stepper feature (Section 10) to get a handle on where exactly your code went wrong.

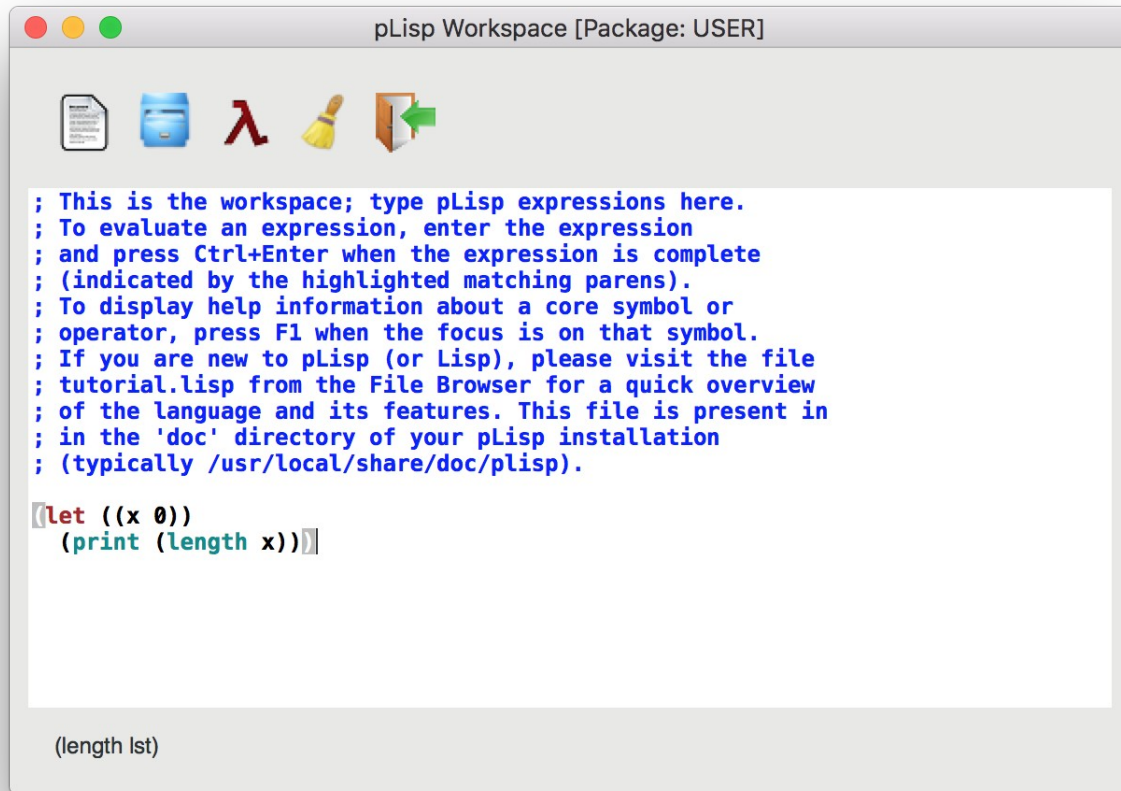


Figure 10

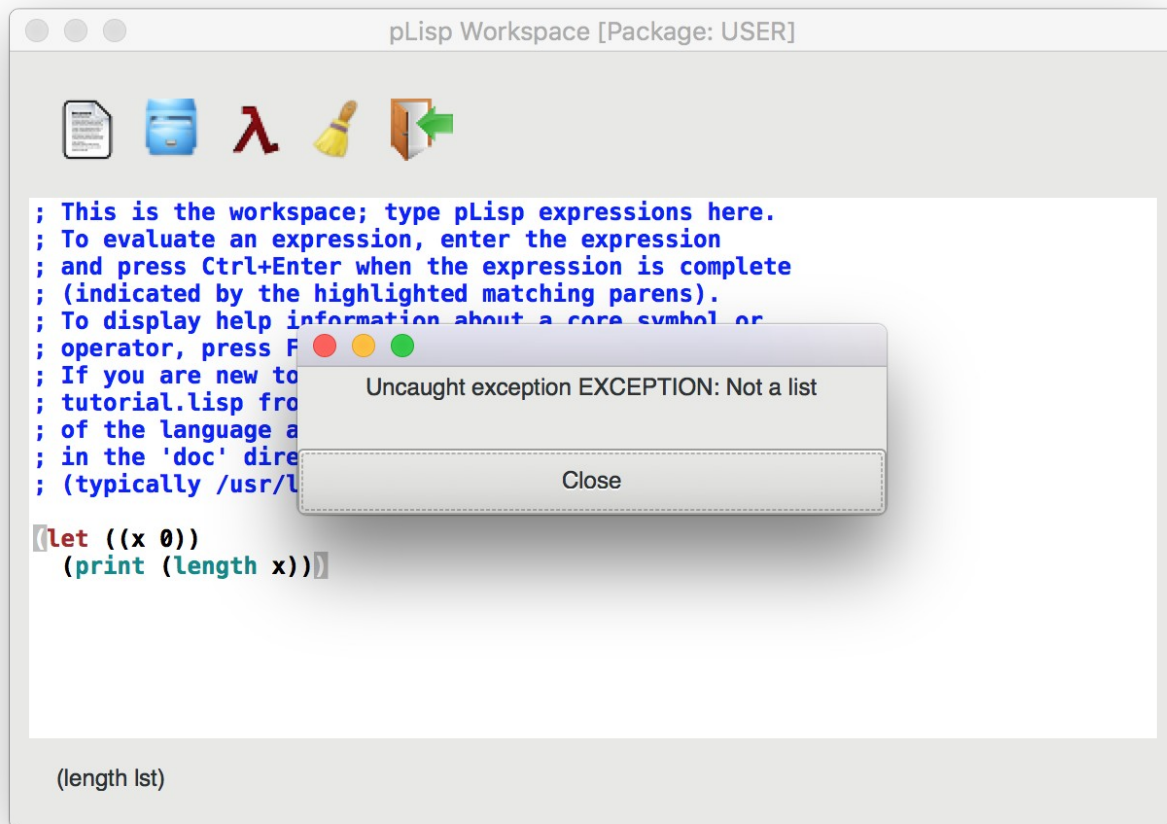


Figure 11

pLisp Debugger	
Function Call	Function Body
(ERROR "Not a list")	(LAMBDA (X) (PRIM-ERROR X))
(NOT NIL)	(LAMBDA (X) (PRIM-NOT X))
(CONSP 0)	(LAMBDA (X) (PRIM-CONSP X))
(EQ 0 NIL)	(LAMBDA (A B) (PRIM-EQ A B))
(NULL 0)	(LAMBDA (X) (EQ X 'NIL))
(LENGTH 0)	(LAMBDA (LST) (IF (NULL LST) 0 (IF (NOT (CONSP LST)) (ERROR "NOT A LIST") (+ 1 (LENGTH (CDR LST))))))
(REPL-FUNCTION)	(LET ((X 0)) (PRINT (LENGTH X)))

Figure 12

13. Online Help

Context-sensitive help is available in pLisp for all the keywords and core forms. Pressing F1 on a symbol when in code-editing mode (i.e. in the Workspace, System Browser code panel or the File Browser; also in the Referrers window) launches the help window for that symbol if available. The status bar in the relevant window also displays the signature of the function or core form when the name of the function or core form is typed. pLisp also comes with an auto-complete feature: if there is no ambiguity about what symbol you're referring to when you press tab midway through the typing of the symbol name, pLisp fills in the remaining part of the name.

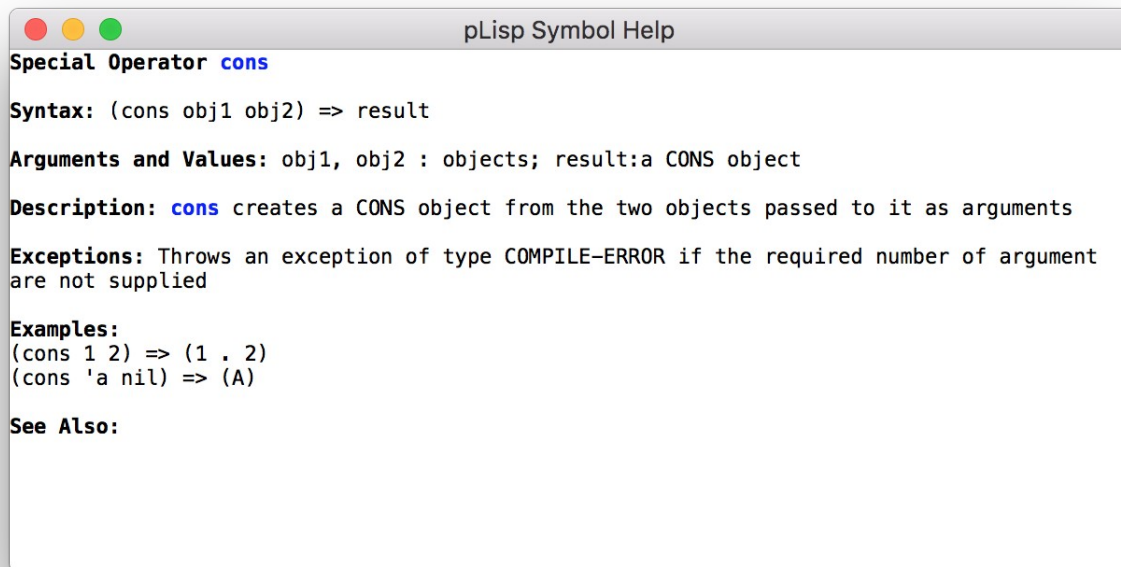


Figure 13

14. Object Inspector

pLisp provides a special form called 'inspect-object' which can be used to more easily visualize objects in the GUI. Atomic objects (literal objects like integers, floats, strings, etc.) are displayed as-is in the Object Inspector, while composite objects are displayed as a hyperlink, and the user can drill down into their contents by placing the cursor on the hyperlink and pressing F3. Pressing Escape closes the current inspector window and returns the user to the parent inspector window if it exists. Figure 14 displays the Object Inspector in action.

15. A Note About Packages

pLisp objects are categorized into namespaces called packages. The CORE package contains all the predefined special operators (CAR, CDR, and so on), and it is not permitted to modify this package by adding or modifying any symbols in it. A package called USER is created by default, and the user is expected to define their code in this package. Additional packages can be created if needed, of course.

The evaluation of any expression is always in the context of the current package, indicated in the title of the Workspace window in GUI mode and in the prompt in the console mode.



Figure 14