LP/LispLite: Trivial Lisp ⇔ Org Mode Conversion

Roy M. Turner

Spring 2016

Contents

1 Introduction

This file is one of LP/Lisp's files. It implements a very trivial conversion between Lisp files and Org Mode (Emacs) files—and I mean *very* trivial. Basically, this will take a Lisp file with Org Mode text in comments and output a properly-formatted Org Mode file, complete with Lisp source in `#+BEGIN_SRC/#+END_SRC` blocks. Alternatively, given an Org Mode file, it will produce a Lisp file with the Org Mode text in comments. The two files will likely **not** look the same, however; there will be extra blank lines, etc., in one or the other (or both). In particular, if you convert a Lisp file to an Org Mode file and then back to Lisp. you'll have a duplicate Org Mode header. . . one for each time you make the conversion back and forth.

There is one other thing that currently is needed to make output from the Org Mode file look good. For some reason, the tabs on the Lisp lines in the Org Mode file aren't getting expanded right. Org Mode (at least the version I have installed, which is 20160215, doesn't work as advertised, and neither the `-i` switch on the `BEGIN_SRC` line nor any variable setting seems to help anything. So for now, you need to explicitly untabify the Org Mode buffer before output (e.g., `M-x untabify` in Emacs). Argh.

If you are interested in a *much* more capable literate programming tool for Lisp, see LP/Lisp itself. Unfortunately, however, it only works at the moment with Allegro Common Lisp!

2 Using the tool

When you load the LP/LispLite Lisp file, it puts all functions in the `lplisp` package. This means that functions and object names, etc., from this file need to be prefaced with `lplisp:`.

To convert from the Org version of your file to the Lisp version, do:

```
(lplisp:convert (make-instance 'lplisp:org-to-lisp :filename "your-file.org"))
```

By default, this will create a file `"your-file.lisp"` in the current directory. If you want the output to go elsewhere, use the `:output-file` keyword parameter.

To convert from the Lisp version of your file to an Org version, do:

```
(lplisp:convert (make-instance 'lplisp:lisp-to-org :filename "your-file.lisp"))
```

By default, this will create a file `"your-file.org"` in the current directory. If you want the output to go elsewhere, use the `:output-file` keyword parameter.

Note: Be careful! These functions will happily overwrite your `.lisp` and `.org` files without warning! So *please* back up your files before you convert them!

3 Header/miscellaneous

This resides in the `lplisp` package, which we have to create if it doesn't exist.

```
1  (unless (find-package "LPLISP")
2    (defpackage "LPLISP"
3      (:use "COMMON-LISP")
4      (:export "LISP-TO-ORG"
5        "ORG-TO-LISP"
6        "CONVERT"
7        )))
```

```
8  (in-package lplisp)
```

Require the pattern matcher

```
 9  (require 'CL-PPCRE)
10  (use-package 'cl-ppcre)
```

4 Variables/parameters

4.1 Variable: =*short-comment-start-regexp*

Set this to what starts short (single-line) comments in the input language. By default, it's set for Lisp. Like other "regexp" variables, this one is initially set to contain a CL-PPCRE "scanner"; this *should* speed up processing.

```
11  (defvar *short-comment-start-regexp* (create-scanner "^\\s*;;;+ ?"))
```

4.2 Variable: =*org-command-start*

This regexp matches the start of Org Mode commands. Note that in going from **to** Org, this will be preceded by the input language's short comment string.

```
12  (defvar *org-command-start* (create-scanner "^\\s*#\\+"))
```

4.3 Variable: =*org-comment-start*

This regexp matches the start of Org Mode comments. Note that in going from **to** Org, this will be preceded by the input language's short comment string.

```
13  (defvar *org-comment-start* (create-scanner "^\\s*#[^+]"))
```

4.4 Variable: `*long-comment-start-regexp*`

This matches the start of along comment. We expect this to be the first thing on the line, other than white space.

```
14  (defvar *long-comment-start-regexp* (create-scanner "^\\s*#\\|"))
```

4.5 Variable: `*long-comment-end-regexp*`

This matches the end of along comment. We expect this to be the *last* thing on a line – anything after this will be deleted.

```
15  (defvar *long-comment-end-regexp* (create-scanner "\\|#"))
```

4.6 Variable: `*lisp-block-begin-regexp*`

This is used to find the `#+begin_src` line that starts a Lisp code block. Since we need this to be case-insensitive, we use a scanner rather than just a regular expression.

```
16  (defvar *lisp-block-begin-regexp* (create-scanner "^\\s*#\\+begin_src\\s+lisp"
17     :case-insensitive-mode t))
```

4.7 Variable: `*lisp-block-end-regexp*`

This is used to find the `#+end_src` line that ending a Lisp code block. Since we need this to be case-insensitive, we use a scanner rather than just a regular expression.

```
18  (defvar *lisp-block-end-regexp* (create-scanner "^\\s*#\\+end_src"
19     :case-insensitive-mode t))
```

4.8 Variable: `*results-line-regexp*`

```
20  (defvar *results-line-regexp* (create-scanner "^\\s*#\\+results"
21         :case-insensitive-mode t))
```

4.9 Variable: `*org-mode-header*`

Header lines to include at the start of the Org Mode file when converting from Lisp.

```
22  (defvar *org-mode-header*
23      '(
24        "# ############################################################################"
25        "# Start: Added by LP/Lisp"
26        "#+STARTUP: hidestars"
27        "#+STARTUP: showall"
28        "#+OPTIONS: toc:t num:t"
29        "#+DATE: "
30        "#+LATEX_CLASS_OPTIONS: [11pt]"
31        "# Fix the margins -- following from Clark Donley (clarkdonley.com)"
32        "#+LATEX_HEADER: \\usepackage[margin=1in]{geometry}"
33        "# This line makes lists work better:"
34        "# It eliminates whitespace before/within a list and pushes it tt the left margin"
35        "#+LATEX_HEADER: \\usepackage{enumitem}"
36        "#+LATEX: \\newpage"
37        "# End: Added by LP/Lisp"
38        "# ############################################################################"
39        )
40    )
```

4.10 Variable =*org-to-lisp-line-number-regexp*

This variable holds a regexp matching the numbers that LP/Lisp puts in front of source (Lisp) lines in the Org Mode file. These are stripped out when importing back into Lisp from an Org File by `org-to-lisp`.

```
41  (defvar *org-to-lisp-line-number-regexp* "^\\s*\[[0-9]*\]\\s?")
```

5 Generic function definitions

This section is here because apparently `initialize-instance` has to have functions it calls defined before it is defined. Go figure. Lisp thinks it's C, maybe?

```
42  (defgeneric change-extension (old-filename new-extension))
```

6 Convert from Lisp to Org Mode: the `lisp-to-org` class

```
43  (defclass lisp-to-org ()
44    (
45     (filename :initform nil :initarg :filename)
46     (output-file :initform nil :initarg :output-file)
47     ;; Streams:
48     (in :initform nil)
49     (out :initform nil)
50     (lineno :initform 0)
51     )
52    )
```

Initialize instance (`:after`) method

This adds functionality to the `initialize-instance` method for `lisp-to-org` to create the output filename from the input filename, if it isn't specified.

```
53  (defmethod initialize-instance :after ((self lisp-to-org) &key &allow-other-keys)
54    (with-slots (filename output-file) self
55      (when filename
56        (unless output-file
57   (setf (slot-value self 'output-file) (change-extension filename ".org"))))))
```

6.1 Method `convert`

This method does all the work for `lisp-to-org`.

```
58  (defmethod convert ((self lisp-to-org) &key (omit-header nil) &allow-other-keys)
59    (with-slots (lineno in out filename output-file) self
60      (with-open-file (instream filename :direction :input)
61        (setq lineno 0)
62        (setq in instream)
63        (with-open-file (outstream output-file :direction :output :if-exists :supersede
64    :if-does-not-exist :create)
65   (setq out outstream)
66   (unless omit-header
67     (dolist (header *org-mode-header*)
68       (format out "~a~%" header)))
69   (let ((line (read-line in nil :eof)))
70     (loop until (eql :eof line)
71         do (setq line (case (line-type self line)
72         (:empty-line (read-line in nil :eof))
73         (:long-comment (do-long-comment self line))
74         (:comment (do-comment-line self line))
75  ;       (:org-command (do-org-command-line self line))
76         (otherwise (do-lisp-lines self line)))))))))
```

6.2 Method `do-comment-line`

This takes care of a comment line by converting it to a plain Org Mode line.

```
77  (defmethod do-comment-line ((self lisp-to-org) line)
78    (with-slots (in out) self
79      (let ((start-char (search ";" line)))
80        (write-line (strip-chars line *short-comment-start-regexp*) out)
81        (read-line in nil :eof))))
```

6.3 Method `do-org-command-line`

[OBSOLETE] This isn't really needed, I don't think?

```
82  (defmethod do-org-command-line ((self lisp-to-org) line)
83    (with-slots (in out) self
84      (write-line (concatenate 'string ";;; " line) out)
85      (read-line in nil :eof)))
```

6.4 Method `do-long-comment`

This handles an entire long comment. It just takes the leading and trailing delimiters off and writes out all other lines directly.

```
86  (defmethod do-long-comment ((self lisp-to-org) line)
87    (with-slots (in out) self
88      ;; Output first line, minus the #| -- in case it wasn't otherwise empty:
89      (write-line (strip-long-comment-start-chars self line) out)
90      (loop until (or (eql :eof (setq line (read-line in nil :eof)))
91       (eql :long-comment-end (line-type self line)))
92   do
93     (write-line line out))
94      (cond
95       ((eql :eof line)
96        :eof)
97       (t (write-line (strip-long-comment-end-chars self line) out)
98   (read-line in nil :eof)))))
```

6.5 Method `do-lisp-lines`

This handles an entire section of Lisp code by wrapping it in a `#+BEGIN_SRC/#+END_SRC` block. It also causes line numbers to be output from the Org file by using Org's `+n` switch to the source block. We also keep track of the line number, though, so we can (later) write an index for the functions, etc.

```
 99  (defmethod do-lisp-lines ((self lisp-to-org) line)
100    (with-slots (in out lineno) self
101      (write-org-begin-src self)
102  ;    (format out "[~5,'0d] ~a~%" (incf lineno) line)
103      (incf lineno)
104      (write-line line out)
105      (loop until (or (eql :eof (setq line (read-line in nil :eof)))
106  ;     (not (member (line-type self line) '(:source-line :empty-line))))
```

```
107       (not (eql :source-line (line-type self line))))
108   do
109  ;   (format out "[~5,'0d] ~a~%" (incf lineno) line)
110      (incf lineno)
111      (write-line line out)
112      )
113       (write-org-end-src self)
114       (if (eql :eof line)
115         :eof
116         line)))
```

6.6 Method `empty-line`

Returns t if the line contains nothing but white space.

```
117  (defmethod empty-line ((self lisp-to-org) line)
118    (tmatch "^\\s*$" line))
```

6.7 Method `line-type`

This returns the type of `line`. It can be one of:

- :comment – a single-line comment
- :long-comment/:long-comment-end – start/finish of a long comment block
- :org-command [obsolete]
- :empty-line – just that
- :source-line – a Lisp (e.g.) source code line

Note: There is a comment convention in play here for Lisp files. Lines with 3 or more semicolons are considered `:comment` lines, and so will be formatted as Org Mode text. Less than this, `:source-line` will be returned as the type, so that in-line comments that don't warrant documentation-level treatment on their own, as well as commented-out functions, can be left in the Lisp source code.

```
119  (defmethod line-type ((self lisp-to-org) line)
120    (cond
121     ((tmatch *short-comment-start-regexp* line) :comment)
122     ((tmatch *long-comment-start-regexp* line) :long-comment)
123     ((tmatch *long-comment-end-regexp* line) :long-comment-end)
124     ((tmatch *org-command-start* line) :org-command)
125     ((tmatch "^\\s*$" line) :empty-line)
126     (t :source-line)))

127  (defmethod write-org-begin-src ((self lisp-to-org))
128    (with-slots (out) self
129      (write-line "#+BEGIN_SRC lisp +n -i :tangle yes :comments link" out)))

130  (defmethod write-org-end-src ((self lisp-to-org))
131    (with-slots (out) self
132      (write-line "#+END_SRC" out)))
```

6.8 Method `strip-comment-chars`

This will strip comment characters from the front of line.

```
133  (defmethod strip-comment-chars ((self lisp-to-org) line)
134    (strip-chars self line *short-comment-start-regexp*))
```

6.9 Method `strip-long-comment-start-chars`

```
135  (defmethod strip-long-comment-start-chars ((self lisp-to-org) line)
136    (strip-chars self  line *long-comment-start-regexp*))
```

6.10 Method `strip-long-comment-end-chars`

Unlike mos of the `strip-xyz` methods, this one strips stuff off the end rather than the beginning. Note that everything after the end regexp is deleted, as it is assumed that this is the last thing on the line.

```
137  (defmethod strip-long-comment-end-chars ((self lisp-to-org) line)
138    (multiple-value-bind (start end)
139        (tmatch *long-comment-end-regexp* line)
140      (declare (ignore end))
141      (cond
142       ((null start) line)
143       (t (subseq line 0 start)))))
```

7 Convert from Org Mode to Lisp: class `org-to-lisp`

This class takes an Org Mode file containing Lisp code and converts it into a Lisp file, with all Org Mode commands, etc., in comments, and with the Lisp source code (in `#+BEGIN_SRC/#+END_SRC` blocks) extracted. This does *not*, unfortunately, do any moving around of code like a true literate programming system (like the full-fledged LP/Lisp) should do. It can handle, however, Lisp lines preceeded by `[XXXXX]` strings (line numbering), and it will remove the leading numbering.

```
144  (defclass org-to-lisp ()
145    (
146     (filename :initform nil :initarg :filename)
147     (output-file :initform nil :initarg :output-file)
148     (in :initform nil :initarg :in)
149     (out :initform nil :initarg :out)
150     (comment-start :initform ";;;" :initarg :comment-start)
151     )
152    )
```

Initialize instance (`:after`) method

This adds functionality to the `initialize-instance` method for `org-to-lisp` to create the output filename from the input filename, if it isn't specified.

```
153 (defmethod initialize-instance :after ((self org-to-lisp) &key &allow-other-keys)
154   (with-slots (filename output-file) self
155     (when filename
156       (unless output-file
157  (setf (slot-value self 'output-file) (change-extension filename ".lisp"))))))
```

7.1 Method `convert`

This does the real work of conversion.

```
158 (defmethod convert ((self org-to-lisp) &key &allow-other-keys)
159   (format t "*short-comment-start-regexp*=~s~%" *short-comment-start-regexp*)
160   (with-slots (in out filename output-file) self
161     (with-open-file (instream filename :direction :input)
162       (setq in instream)
163       (with-open-file (outstream output-file :direction :output :if-exists :supersede
164         :if-does-not-exist :create)
165  (setq out outstream)
166  (let ((line (read-line in nil :eof)))
167    (loop until (eql :eof line)
168        do (setq line (case (line-type self line)
169        (:begin-src (extract-lisp-code self line))
170        (otherwise  (comment-org-mode-lines self line)))))))))
```

7.2 Method `extract-lisp-code`

This will handle a source code block by getting rid of the delimiters and any leading line numbers of the form `[xxxxxx]`.

```
171 (defmethod extract-lisp-code ((self org-to-lisp) line)
172   (with-slots (in out) self
173     ;; Skip the =BEGIN_SRC= command by reading the file again immediately at
174     ;; the head of the loop.  However, I should put a blank line in its place.
175     (write-line "" out)
176     (loop with start
177  with end
178  until (or (eql :eof (setq line (read-line in nil :EOF)))
179    (eql :end-src (line-type self line)))
180  do
181    (multiple-value-setq (start end)
182      (tmatch *org-to-lisp-line-number-regexp* line))
183    (cond
184     ((null start) (write-line line out))
185     (t
186      (write-line (subseq line end) out))))
187     (write-line "" out)
188     ;; If not at the end of the file, then we've encountered an =END_SRC_
189     ;; command; read another line to skip it.
190     (unless (eql line :eof)
```

```
191         (setq line (read-line in nil :eof)))
192       line))
```

7.3 Method `comment-org-mode-lines`

This inserts non-source lines as comment lines in the Lisp file.

```
193  (defmethod comment-org-mode-lines ((self org-to-lisp) line)
194    (with-slots (in out comment-start) self
195      (write-line comment-start out)
196      (loop until (or (eql :eof line)
197       (eql :begin-src (line-type self line)))
198   do
199     (cond
200      ((eql :results-line (line-type self line))
201       ;; Skip the results, if any:
202       (setq line (skip-results-line self line)))
203      (t
204       (write-line (concatenate 'string comment-start " " line) out)
205       (setq line (read-line in nil :eof)))))
206      line))
```

```
207  (defmethod skip-results-line ((self org-to-lisp) line)
208    (with-slots (in) self
209      (format t "Skipping ~s~%" line)
210      (loop until (or (eql :eof (setq line (read-line in nil :eof)))
211       (not (tmatch "^\\s*: " line)))
212   do (format t "Skipping ~s" line))
213      line))
```

7.4 Method `line-type`

Similar to the corresponding method for class `lisp-to-org`; the valid types are `:org-mode-line` and `:begin/end-src`.

Note that we have to bind `*package*` to the `lplisp` package for the duration of this method, since otherwise `read-from-string` will intern the thing read into whatever package the *user* is in, not `lplisp`. I know this is how it is supposed to work, but it sure seems like a poor design choice to me. (Or maybe it's just due to `lplisp` "using" `cl-user`?)

```
214  (defmethod line-type ((self org-to-lisp) line)
215    (let ((*package* (find-package 'lplisp)))
216      (cond
217       ((tmatch *lisp-block-begin-regexp* line)
218        :begin-src)
219       ((tmatch *lisp-block-end-regexp* line)
220        :end-src)
221       ((tmatch *results-line-regexp* line)
222        :results-line)
223       (t
224        :org-mode-line))))
```

8 Utility functions

8.1 Method `change-extension(old-filename new-extension)`

```
225 (defmethod change-extension ((self string) new)
226   (let ((period (search "." self :from-end t)))
227     (concatenate 'string (if period
228     (subseq self 0 period)
229     self)
230   new)))
```

8.2 Method `tmatch` (of string): trivial match function

This is just a wrapper around CL-PPCRE's `scan` function for now. It also returns what `scan` returns (i.e., 4 values). Regexp can be a scanner or a regexp.

```
231 (defmethod tmatch ((regexp string) (string string) &key (start 0) (end (length
232  string)))
233   (scan regexp string :start start :end end))

234 (defmethod tmatch (regexp (string string) &key (start 0) (end (length
235  string)))
236   (scan regexp string :start start :end end))
```

8.3 Method `strip-chars` (of string)

This is called by most of the other `strip-xyz` methods to do the bulk of the work. It strips off the leading portion of `string` that matches `regexp`.

```
237 (defmethod strip-chars ((line string) regexp)
238   (multiple-value-bind (start end)
239       (tmatch regexp line)
240     (cond
241      ((null start)
242       line)
243      (t
244       (subseq line end)))))

245 (defclass python-to-org (lisp-to-org)
246   ()
247   )

248 (defmethod convert :around ((self python-to-org) &key (omit-header nil) &allow-other-keys)
249   (let (
250  (*short-comment-start-regexp* (create-scanner "^\\s*#+ ?"))
251  (*long-comment-start-regexp* (create-scanner
252        (concatenate 'string '(#\" #\" #\"))))
253  (*long-comment-end-regexp* (create-scanner
254      (concatenate 'string '(#\" #\" #\"))))
255  (*lisp-block-begin-regexp* (create-scanner "^\\s*#\\+begin_src\\s+python"
```

```
256      :case-insensitive-mode t)))
257      (call-next-method)))

258  (defclass org-to-python (org-to-lisp)
259    ((comment-start :initform "##"))
260    )

261  (defmethod initialize-instance :after ((self org-to-python) &key &allow-other-keys)
262    (with-slots (filename output-file) self
263      (when filename
264        (unless output-file
265   (setf (slot-value self 'output-file) (change-extension filename ".py"))))))

266  (defmethod convert :around ((self org-to-python) &key &allow-other-keys)
267    (let (
268   (*short-comment-start-regexp* (create-scanner "^\\s*#+ ?"))
269   (*long-comment-start-regexp* (create-scanner
270         (concatenate 'string '(#\" #\" #\"))))
271   (*long-comment-end-regexp* (create-scanner
272       (concatenate 'string '(#\" #\" #\"))))
273   (*lisp-block-begin-regexp* (create-scanner "^\\s*#\\+begin_src\\s+python"
274      :case-insensitive-mode t)))
275      (call-next-method)))
```