Emacs Lisp Regression Testing (ERT) Reference Card

Fabrice Niessen

2015-08-17

Contents

1. Documentation

ERT manual[1]

[1] http://www.gnu.org/software/emacs/manual/ert.html

2. How to run tests

2.1 Emacs tests

2.1.1 Running tests interactively

`ert-run-tests-interactively` runs tests and offers an interactive interface for inspecting results and debugging.

`M-x ert RET t RET` Runs **all** the **tests** currently defined and display the results in a buffer.

2.1.2 Running tests in batch mode

There is also `ert-run-tests-batch-and-exit` for non-interactive use.

To run the tests in batch mode, type:

```
# Run all defined tests (no selection).
emacs --batch -l my-tests.el -f ert-run-tests-batch-and-exit
emacs -Q --batch -L . -l yasnippet-tests.el -nw -e yas-batch-run-tests
```

2.2 Org tests

2.2.1 Running tests interactively

```
(require 'org-test)
```

`M-x org-test-run-all-tests` Runs all the tests currently defined matching the string `\(org\|ob\)`.

Unlike `org-test-run-all-tests`, `ert` does not "touch" the examples, what avoids confirmation on the following runs.

To run **each test individually**, press `C-x C-e` at the end of the `should` sexp ("S-expression").

2.2.2 Running tests in batch mode

To run the test suite (**all existing test cases**), type `make test` from the `org-mode/` directory.

```
# Using emacs in batch mode (with no user and site configuration).
emacs -Q --batch \
      --eval '(setq vc-handled-backends nil org-startup-folded nil)' \
      --eval '(add-to-list '"'"'load-path (concat default-directory "lisp"))' \
      --eval '(add-to-list '"'"'load-path (concat default-directory "testing"))' \
      -l org-batch-test-init \
      --eval '(setq \
                    org-batch-test t \
                    org-babel-load-languages \
                    org-test-select-re "\\(org\\|ob\\)" \
              )' \
      -l org-loaddefs.el \
      -l cl -l testing/org-test.el \
      -l ert -l org -l ox \
      --eval '(org-test-run-batch-tests org-test-select-re)'

# Run all defined tests (no selection).
emacs -Q --batch \
      -L lisp/ -L testing/ \
      -l ~/src/emacs-leuven/org-test-fni.el \
      --eval '(setq org-confirm-babel-evaluate nil)' \
      -f ert-run-tests-batch-and-exit
```

2.3 Key bindings

Key bindings in the ERT results buffer.

2.3.1 Running

R **Re-runs all tests**, using the same selector.

r **Re-runs the test** near point.

h Displays the **documentation** of the test at point.

. Jumps to the **definition** (source code) of the test.

2.3.2 Navigation

j **Jumps** back and forth **between** the test run **summary and** individual test **results**.

n Moves point to the **next** test **result**.

p Moves point to the **previous** test **result**.

q **Quits** window and bury its buffer.

2.3.3 Profiling

T Display test **timings** for the last run.

3. How to debug tests

Interactive debugging in the ERT buffer.

d Re-runs the test with the **debugger** enabled.

b On a failed test, shows the **backtrace** of the failure. More convenient than `d` as it strips out the irrelevant top few frames shown in the debugger backtrace.

l If the test contains a series of `should` forms, shows the **list** of all `should` forms executed during the test before it failed.

m Shows **messages** which were printed for the test before it failed.

L On a failed test, increases the "printer" limits (`print-length` and `print-level`) to show more of the expression.

D Lets ERT **forget** about the obsolete test at point (because edited and rearranged).

4. How to write tests

4.1 Macros

Operators available:

should Check that the assertion is true.

should-not Check that the predicate returns `nil`.

should-error Check that the form called within it signals an error.

skip-unless Skip the test immediately without processing further. This is useful for checking the test environment (like availability of features, external binaries, etc).

I suggest to always put the `should` (or `should-not`, `should-error`) outside each test: it makes it easier to inspect results from partial evaluations.

4.2 Create a new test case

To write your first test:

- Find a similar test in `testing/lisp/<file>.el` (maybe in `test-ob-exp.el`)
- Load the file `testing/org-test.el` which has many helper functions
- Write and evaluate your `ert-deftest`:

```
(ert-deftest test-org/end-of-line ()
  "Test `org-end-of-line' specifications."
  ;; At an headline without special movement.
  (should
   (org-test-with-temp-text "* Headline2 :tag:\n"
     (let ((org-special-ctrl-a/e nil))
       (and (progn (message "FOO")
                   (org-end-of-line)
                   (message (format "%d" (point)))
                   (eolp))
            (progn (org-end-of-line)
                   (eolp)))))))
```

- Call `ert` to run the test

```
(ert 'test-org/end-of-line)
```

4.3 Useful tips and tricks

4.3.1 Position of point

When `org-babel-execute-src-block` is called, the **point must be inside of the code block**. Hence, search forward in the text to place the point at the beginning of the code block before executing!

4.3.2 RESULTS vs results

For the string equality to return `true`, you have to uppercase `RESULTS`. In the future, more flexible tests (such as regular expression searches) rather than strict equality should be preferable.

Or you could let-bind `org-babel-results-keyword` to `"results"` around any future tests.

4.3.3 Duplication of input

> I'm thinking at another thing that could help reduce the size of the
> tests. Currently, we copy once the code block, and once the code block + its
> results.
>
> Maybe we could have a function to locate the results, and only check on the
> results.
>
> So, something along those lines:
>
> –8<—————cut here—————start—————->8—
> (should
> (equal (results-part (org-babel-execute-src-block "code block only"))
> "results only"))
> –8<—————cut here—————end—————->8—
>

> No duplication of the input…
>
> Does this make sense?

Yes, that would be an improvement, see the other tests in that file for examples of similar functionality.

4.3.4 Temporary buffer

Use a **test buffer** (with Org mode enabled) to avoid messing with test example.

Use the function `org-test-with-temp-text` to use a **temporary Org mode buffer with initial text**.

```
(org-test-with-temp-text "Initial text"
  ...)
```

It is then possible that the test simply compares the whole `buffer-string` with some expected contents.

`(kill-buffer)` within `(with-temp-buffer ...)` does not make sense.

5. Extending Org tests

These are several examples which you can use as patterns to add tests to Org.

5.1 Important functions

Functions to look at.

5.1.1 Ob.el

```
org-babel-execute-src-block
org-babel-execute:<language>
org-babel-expand-body:generic
org-babel-get-header
org-babel-get-inline-src-block-matches
org-babel-get-src-block-info
org-babel-next-src-block
```

5.1.2 Org-test.el

```
org-test-at-id
org-test-in-example-file
org-test-with-temp-text
```

5.1.3 Subr.el

```
dotimes
match-string
```

5.1.4 C source code

```
mapcar
```

5.2 Insert a new heading

```
(ert-deftest test-org/insert-heading ()
  "Test specifications for heading insertion."
  ;; At the end of a single headline: Create headline below, following
  ;; `org-blank-before-new-entry' specifications.  When it is `auto',
  ;; since there's not enough information to deduce what is expected,
  ;; create it just below.
  (should
   (equal "* H\n* "
          (org-test-with-temp-text "* H"
            (end-of-line)
            (let ((org-blank-before-new-entry '((heading . nil))))
              (org-insert-heading))
            (buffer-string))))
  (should
   (equal "* H\n\n* "
          (org-test-with-temp-text "* H"
            (end-of-line)
            (let ((org-blank-before-new-entry '((heading . t))))
              (org-insert-heading))
            (buffer-string))))
  (should
   (equal "* H\n* "
          (org-test-with-temp-text "* H"
            (end-of-line)
            (let ((org-blank-before-new-entry '((heading . auto))))
              (org-insert-heading))
            (buffer-string))))
  ;; Etc.
  )
```

5.3 Check for compile error

```
  (set-buffer (get-buffer-create "*lilypond*"))
  (erase-buffer)
  (insert-file-contents "babel.sh")
  (goto-char (point-min))
  (search-forward "error:")
```

5.4 Export

```
     (let ((html (org-export-as-html nil nil nil 'string 'body-only)))
       ;; check the location of each exported number
       (with-temp-buffer
     (insert html)
         (goto-char (point-min))
     ;; 0 should be on a line by itself
     (should (re-search-forward "0" nil t))
```

5.5 Insert text for testing visual line mode

```
  ;; Standard test with `visual-line-mode'.
  (should
   (org-test-with-temp-text
       "A long line of text\nSome other text"
     (progn (forward-char 2) (cl-dotimes (i 1000) (insert "very "))
        (visual-line-mode 1) (goto-char (point-min)) (org-end-of-line)
        (thing-at-point-looking-at "very"))))
```

5.6 Proper error message

The following test asserts that, when there is a variable without default value, a proper error message is given; at the beginning, the error was much less understandable:

```
Wrong type argument: consp, nil
```

```
(ert-deftest test-org-babel/no-defaut-value-for-var ()
  "Test that the absence of a default value for a variable DOES THROW
  a proper error."
  (org-test-at-id "f2df5ba6-75fa-4e6b-8441-65ed84963627"
    (org-babel-next-src-block)
    (let ((err
       (should-error (org-babel-execute-src-block) :type 'error)))
      (should
       (equal
        '(error "variable \"x\" in block \"carre\" must be assigned a default value")
        err)))))
```

5.7 Speed commands

It looks like these `self-insert-command` functions are special cases. They don't look to their arguments to see what key-press invoked them, but rather they call the `this-command-keys` function for this purpose. We can force the behavior we want by overriding the definition of this function locally, taking this approach the following test case worked for me.

```
(ert-deftest ob-tangle/speed-command-r ()
  "Test that speed command `r' does demote the headline."
  (org-test-with-temp-text "* Speed command"
    (flet ((this-command-keys () "r")) (org-self-insert-command ?r))
    (goto-char (point-min))
    (should (looking-at "\\*\\* Speed command"))))
```

XXX To simulate the press of key, maybe use this:

```
      ;; It's more or less a convention that each language mode binds its
      ;; symbol completion command to `M-TAB' which is a reserved hot key
      ;; under Windows. Way to solve this: when you hit `C-TAB', the command
      ;; normally bound to `M-TAB' will be called.
      (global-set-key
       (kbd "<C-tab>") '(lambda ()
                          (interactive)
                          (call-interactively (key-binding (kbd "M-TAB")))))
```

5.8 Results block

```
(ert-deftest test-org-babel/just-one-results-block ()
  "Test that evaluating two times the same code block does not result in a
duplicate results block."
  (org-test-with-temp-text "#+begin_src sh\necho Hello\n#+end_src\n"
    (org-babel-execute-src-block)
    (org-babel-execute-src-block) ;; second code block execution
    ;; where is point (supposed to be)?
    (goto-char (point-min))
    (should (search-forward "Hello")) ;; the string inside the source code block
    (should (search-forward "Hello")) ;; the same string in the (first?) results block
    (should-error (search-forward "Hello"))))
```

5.9 Test Org list

```
(ert-deftest org-list-item-test ()
  (with-temp-buffer
    (org-mode)
    (let ((org-allow-alphabetical t)
          (fill-column 70))
      (insert "1. some stuff\n"
              "    a) an alphabetic list item with text longer that the current fill column
      (fill-paragraph)
      (should (not (equal (org-in-item-p) 1))))))
```

6. Contributing

6.1 Issues

Report issues and suggest features and improvements on the GitHub issue tracker[2] .

6.2 Patches

I love contributions! Patches under any form are always welcome!

6.3 Donations

If you like the refcard-ERT project, you can show your appreciation and support future development by making a donation[3] through PayPal.

Regardless of the donations, refcard-ERT will always be free both as in beer and as in speech.

7. License

[2]https://github.com/fniessen/refcard-ERT/issues/new
[3]https://www.paypal.com/cgi-bin/webscr?cmd=_donations&business=VCVAS6KPDQ4JC&lc=BE&item_number=refcard%2dERT&currency_code=EUR&bn=PP%2dDonationsBF%3abtn_donate_LG%2egif%3aNonHosted

Author: Fabrice Niessen
Keywords: ert emacs regression testing