My Lisp Unit Test Framework

Gene Michael Stover

created Friday, 2005 June 17
updated Wednesday, 2005 December 28

Copyright © 2005 Gene Michael Stover. All rights reserved. Permission to copy, store, & view this document unmodified & in its entirety is granted.

Contents

1 What is this?

This is a description of the unit test framework I use for Lisp. It's called CyberTiggyr Test. Anyone is welcomed to use it, but this essay & source code are here mostly for my own use.

1.1 Other Unit Test Frameworks for Lisp

Mine is hardly the only one. There are also...

- Chris Riesbeck's lisp-unit[1]
- Hierarchical Extensible Unit Testing Environment for Common Lisp[2] (HEUTE)
- probably many others.

2 Examples

Here are some example tests.[3]

```
(load "test.lisp")
(import 'cybertiggyr-test:deftest)

(deftest test0000 ()
  "Null test.  Always succeeds.  Usefull when everything is
going wrong."
  'test0000)

(deftest test0010 ()
  "Ensure we can create a circle.  Use default values.  Do
not check that the circle is valid,just that we can create
one."
  (make-circle))

(deftest test-curvature ()
  (let ((circle (make-circle :radius 3)))
    (unless (= (/ (circumference circle) (radius circle) pi))
      ;; In a spherical field, pi has a different meaning...or
      ;; value, depending on whom you ask.
      (format t "~&We're not in Kansas any more."))
    (= (/ (circumference circle) (radius circle) pi))))

(deftest test0100 ()
```

[1] http://www.cs.northwestern.edu/academics/courses/325/readings/lisp-unit.html
[2] http://www.rdrop.com/users/jimka/lisp/heute/heute.html
[3] Examples should come early in the documentation for a library so you don't have to read the whole thing to figure out if the library is the kind of library you want.

```
  "Ensure the config file exists & we have permission to read it."
  (let* ((pathname (make-pathname :name "config" :type "lisp"))
         (is-good (with-open-file (strm pathname
                                        :if-does-not-exist nil)
                    (and strm (read strm)))))
    (unless is-good
      (format t "~&Could not open ~S for input." pathname)
      (format t "~&Does it exist?  Do we have permission to read it?"))
    is-good))

(cybertiggyr-test:run)                  ; run all the tests
```

3 License

CyberTiggyr Test is licensed according to the Gnu Lesser General Public License[2].

4 Obtaining

The source code is in a single file at http://cybertiggyr.com/gene/lut/test.lisp. It is also in Appendix D.

5 Basic Usage

1. LOAD `test.lisp` into your Lisp environment, as part of the application you are developing, just like you'd load other source files.

   `test.lisp` does not depend on anything other than Common Lisp, so you can load it early.[4]

2. Find a package in which to put some test programs (functions). Lately, I've been putting test programs in the packages they test & leaving them there even for the production code.

3. Once you find the location for some test programs, import the symbol CYBERTIGGYR-TEST:DEFTEST.

   You might do this with IMPORT or with the :IMPORT-FROM or :USE options of DEFPACKAGE. Or you might choose not to import the DEFTEST symbol & always qualify it with the CYBERTIGGYR-TEST package name. Whatever floats your boat.

4. To define a test, use DEFTEST. It works just like DEFUN.

5. To run tests, evaluate `(cybertiggyr-test:run)`. This will execute all the tests you have defined in any package, not just the current one, until

[4] Load early & often.

all tests have passwed or some test fails. It will print progress & error messages to *STANDARD-OUTPUT* .

6 Rules for tests

This is the basic contract to which a test must adhere.

1. A test is a function of no arguments.
2. A test is a function for which CYBERTIGGYR-TEST:TEST-FUNCTION-P returns true. In other words:
   (a) It does *not* have a CYBERTIGGYR-TEST:DISPOSITION property with a value of CYBERTIGGYR-TEST:NOT-A-UNIT-TEST, and
   (b) It's name[5] begins with the characters "TEST", or it has a property CYBERTIGGYR-TEST:DISPOSITION with a value of CYBERTIGGYR-TEST:IS-A-UNIT-TEST.
3. A test returns NIL if & only if it fails. If the test succeeds, it returns true.[6]
4. The test framework will print the fact that a test passes or fails, but if a test should print a diagnostic message about *what* went wrong, the test itself must print that message.
5. The function CYBERTIGGYR-TEST:RUN has control over the order in which tests are run.
6. It is not necessary to export a test from its package. The test may be bound to a private symbol.

7 Suggestions for tests

- A test should print nothing if it succeeds.

  When you have a lot of tests, it is not feasible for every test to print a comprehensive performance & statistics dump which you are expected to read to determine whether the test succeeded for failed. When you have a lot of tests, you don't care about statistics; you just want to know what test failed. The RUN function from CYBERTIGGYR-TEST prints the name of each test as it executes the test, & that's all you need or want when the test succeeds. So a test prints nothing if it succeeds.

- I've been placing test functions in the packages they test. I do not remove the tests from production code. They end up compiled into the production binaries we deliver.

[5] More precisely: the name of the symbol bound to it...
[6] "True" being "not NIL".

- I number the test functions, like this: TEST0000, TEST0010, ... TEST0101. I do the same thing for test programs in other languages.

  At first, it might seems like the name of a test program should indicate what the test tests. In practice, you have so many test programs that making a unique, descriptive name for each is difficult. If a test passes, it doesn't matter what it's called, & if it fails, you'll need to look at its source code, so again, it doesn't matter what it's called.

- Use the CHECK macro as an easy way to evaluate an expression & print it if it fails. Here's an example:

```
(and (check (= (woowoo nil) 1))
     (check (= (woowoo '(1 2 3)) 42)))
```

8 Expert-level details

- To tell CYBERTIGGYR-TEST that a particular function is not a test in spite of its name, give it a DISPOSITION of NOT-A-TEST. For example, to ensure TEST-HOOEY is not a test function, you would evaluate "`(setf (get 'test-hooey 'cybertiggyr-test:disposition) 'cybertiggyr-test:not-a-unit-test)`".

- Conversely, to make CYBERTIGGYR-TEST treat a function as a test in spite of the function's name, give it a DISPOSITION of IS-A-UNIT-TEST, like this: "`(setf (get 'foo 'cybertiggyr-test:disposition) 'cybertiggyr-test:is-a-unit-test)`".

  If you use DEFTEST to define your test functions, you won't need to do this.

- To exclude all functions in a package from being treated as tests, push the package object[7] onto the list CYBERTIGGYR-TEST::*EXCLUDED-PACKAGES*.[8]

  EXCLUDED-PACKAGES is initialized to include SYSTEM & COMMON-LISP.

- By default, functions whose names begin with "TEST" are treated as tests. The prefix "TEST" isn't hard-coded; it's in the variable CYBERTIGGYR-TEST::*PREFIX*. You can change the "TEST" prefix by binding another prefix to CYBERTIGGYR-TEST::*PREFIX*.

- Function CYBERTIGGYR-TEST::TEST-FUNCTION-P determines whether a symbol is bound to a function that should be executed as a test. It uses the afore-mentioned *EXCLUDED-PACKAGES* & *PREFIX* variables and the DISPOSITION property.

[7] I mean the actual package object, not just its name. Given the package name, you can obtain the package object with FIND-PACKAGE .

[8] Note the double colons. CYBERTIGGYR-TEST::*EXCLUDED-PACKAGES* is not exported.

- You can use DEFUN to define tests if the function's name begins with "TEST" or you give it a DISPOSITION property of IS-A-UNIT-TEST.

  Using DEFTEST is safer, though, because it as an API is independant of how functions are identified as tests.

9 History

I think I first wrote CyberTiggyr Test some time in about 2000, though it could have been as late as 2001 October when I wrote a genetic algorithm library called Evie. It went through a re-write which changed the API in 2002 or 2003. The current version is 3, & its API is backward-compatible with the previous one.

A Performance Testing

The "test" in "performance testing" is a misnomer, but I've included a couple of functions for doing it in CyberTiggyr Test. They are in CyberTiggyr Test for convenience.

1. The RATE function tells how fast a function runs. Call it with the function whose rate you want to know; the function must require no arguments. RATE returns a list of three elements. The list's FIRST is calls per second, it's SECOND is the number of times it was called, & it's THIRD is the number of seconds the test ran. All three numbers are positive, but they may be integers, ratios, or floating point depending on details of your Lisp.

2. The RATETABLE function runs RATE for a bunch of functions & writes the results to a stream as a LaTeX table. RATETABLE requires one argument, which is a list. Each element in the list has two elements. It's FIRST is the name of the function to time, & it's SECOND is the functio to time.

B Possible Improvements

Here are some ideas for improvements.

1. Give user control over the order in which tests are run. Maybe a dependency scheme? Whatever the solution, *must* be ignorable when user does not care about the order in which tests run.

2. Export *EXCLUDED-PACKAGES* & *PREFIX*???

3. Should *PREFIX* be a list of prefixes, not just a single prefix?

C Other File Formats

- This document is available in multi-file HTML format at http://cybertiggyr.com/gene/lut/.
- This document is available in Pointless Document Format (PDF) at http://cybertiggyr.com/gene/lut/lut.

I write almost all of my documents in LaTeX ([5], [3]). I compile to PDF with `latex`, `dvips`, & `ps2pdf`. I compile to HTML with `latex2html` ([1], [4]).

D The source code: `test.lisp`

```
;;;
;;; $Header: /home/gene/library/website/docsrc/lut/RCS/lut.tex,v 395.1 2008/04/20 17:25:47 g
;;;
;;; Copyright (c) 2005 Gene Michael Stover.  All rights reserved.
;;;
;;; This program is free software; you can redistribute it and/or modify
;;; it under the terms of the GNU Lesser General Public License as
;;; published by the Free Software Foundation; either version 2 of the
;;; License, or (at your option) any later version.
;;;
;;; This program 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 Lesser General Public License for more details.
;;;
;;; You should have received a copy of the GNU Lesser General Public
;;; License along with this program; if not, write to the Free Software
;;; Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301
;;; USA
;;;

(defpackage "CYBERTIGGYR-TEST"
  (:use "COMMON-LISP")
  (:export
   "*EXCLUDED-PACKAGES*"
   "*PREFIX*"
   "DEFTEST"
   "DISPOSITION"
   "IS-A-UNIT-TEST"
   "NOT-A-UNIT-TEST"
   "RUN"
   "TEST-FUNCTION-P"
   "TEST-FUNCTIONS"))

(in-package "CYBERTIGGYR-TEST")
```

```
;;;
;;; unexported helper functions & stoof
;;;

(defun make-failed-test-p (max strm)
  "Return a predicate which runs a test & tells whether it failed.
The predicate also prints a status to the character output stream
STRM."
  (let ((i 0))
    #'(lambda (test)
;; Show which test we're about to run & what percentage
;; of the test suit has been run.
(format strm "~&~3D% ~A =>" (round (* (/ (incf i) max) 100))
(symbol-bigname test))
(finish-output strm)
(let ((is-good (funcall test))) ; run the test
  ;; Show the test's result.
  (format strm " ~A" (if is-good "good" "FAILED"))
  (not is-good)))))             ; compliment the result

(defun symbol-name-starts-with (symbol starts-with)
  "Return true if & only if the name of the symbol begins with
the string bound to STARTS-WITH."
  (let ((len (length starts-with)))
    (and (>= (length (symbol-name symbol)) len)
 (equal (subseq (symbol-name symbol) 0 len) starts-with))))

(defun symbol-bigname (symbol)
  "Return, as a string, the package name of the symbol & the name
of the symbol."
  (format nil "~A::~A" (package-name (symbol-package symbol)) symbol))

;;;
;;; You could alter these values to fine-tune the behaviour of
;;; TEST-FUNCTION-P.  Adding packages to *EXCLUDED-PACKAGES* is
;;; safe, but altering *PREFIX* could be trouble.
;;;

(defvar *prefix* "TEST" "String prefix of test function names.")

(defvar *excluded-packages*
  (remove (find-package "COMMON-LISP-USER") (list-all-packages))
  "Packages whose functions are not eligible to be test functions.
Defaults to the packages that were loaded before this package, less
COMMON-LISP-USER.")
```

```
(defun test-function-p (symbol)
  "Return true if & only if SYMBOL is bound to a test function."
  (and (fboundp symbol)
       (not (eq (get symbol 'disposition) 'not-a-unit-test))
       (not (member (symbol-package symbol) *excluded-packages*))
       (or (eq (get symbol 'disposition) 'is-a-unit-test)
   (symbol-name-starts-with symbol *prefix*))))
(setf (get 'test-function-p 'disposition) 'not-a-unit-test)

(defun test-functions ()
  "Return a list of symbols bound to test functions in any package."
  (let ((lst ()))
    (do-all-symbols (symbol)
      (when (test-function-p symbol) (push symbol lst)))
    (remove-duplicates (sort lst #'string-lessp :key #'symbol-bigname))))

(setf (get 'test-functions 'disposition) 'not-a-unit-test)

(defun run (&optional (strm *standard-output*))
  "Run all unit tests.  Print results to STRM.  Return true if & only
if all tests pass."
  (let ((max (length (test-functions))))
    (loop for i from 0 to max
          for symbol in (test-functions) do
          (format strm "~&~2D ~A =>" (round (* (/ i max) 100))
  (symbol-bigname symbol))
  (cond ((funcall symbol) (format strm " good"))
 (t (format strm " FAILED")
    (return-from run nil)))))
  'run)

(defun run (&optional (strm *standard-output*))
  "Run all unit tests.  Print results to STRM.  Return true if & only
if all tests pass."
  (null
   (find-if
    ;; Search for a test function which fails...
    (make-failed-test-p (length (test-functions)) strm)
    ;; ...from the suite of test functions.
    (test-functions))))

(defmacro deftest (name &rest body)
  "Declare a unit test function.  For now, maps to DEFUN, but could
be implemented differently in the future."
  (if (symbol-name-starts-with name *prefix*)
```

```
      `(defun ,name ,@body)
    ;; else, We'll need to set DISPOSITION
    `(progn (setf (get ',name 'cybertiggyr-test:disposition)
  'cybertiggyr-test:is-a-unit-test)
    (defun ,name ,@body))))

;;; --- end of file ---
```

References

[1] Nikos Drakos. latex2html.

[2] Free Software Foundation. Lesser general public license. world wide web. http://www.gnu.org/licenses/licenses.html#LGPL.

[3] Michel Goossens and Frank Mittelbach. *The LaTeX Companion*. Addison Wesley Longman, Inc., 1993. ISBN 0201541998.

[4] Michel Goossens and Sebastian Rahtz. *The LaTeX Web Companion: Integrating TeX,* HTML*, and* XML. Addison Wesley Longman, Inc., 1999. ISBN 020143317.

[5] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, Inc., 1986. ISBN 0-201-15790-X.