Multi-File Code Coverage in DrRacket

Jonathan Walsh

Computer Science Department
College of Engineering
California Polytechnic State University
2011

Submitted: May 27, 2011
Advisor: John Clements

Contents

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

List of Figures

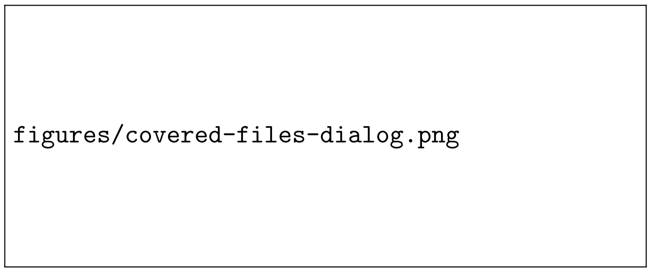

Figure 1: Covered Files Dialog

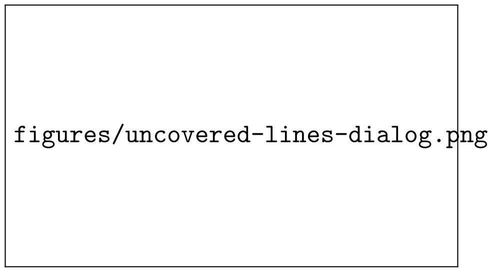

Figure 2: Uncovered Lines Dialog

I Introduction

Code coverage is widely used software testing tool [**?**].

II Background

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

III Implementation

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

IV Evaluation

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

V Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

`figures/no-coverage-found-dialog.png`

Figure 3: No Coverage Found Dialog

`figures/out-of-date-dialog.png`

Figure 4: Out Of Date Dialog

`figures/coverage-button.png`

Figure 5: Multi-File Coverage Button in DrRacket

A Source Code

info.rkt

```
#lang setup/infotab
(define name "Multi-File Code Coverage")

(define drracket-tools (list (list "tool.rkt")))
(define drracket-name name)

(define multi-file-code-coverage-info-file #t) ;used by info-helper to find this file

;Names, labels, and other items that the docs and source code will have in common
(define tool-name name)
(define button-label "Multi-File Coverage")
(define open-with-label "Open With Uncovered Lines Dialog")
(define coverage-suffix ".rktcov")


;Stuff for PLanet
(define blurb
  `("Extends code coverage highlighting to multiple files"))
(define categories `(devtools))
(define primary-file '("main.rkt"))
(define release-notes
  `("Updated Documentation"))
(define version "0.4")
(define repositories '("4.x"))
(define scribblings '(("code-coverage.scrbl" ())))
```

info-helper.rkt

```
#lang racket
;Provides the names of labels and other items that are defined in info.rkt and used across
; tool.rkt and code-coverage.scrbl so that to change the value in all 3 locations only
; info.rkt needs to be updated
(require setup/getinfo)
(require syntax/location)
(provide info-look-up
         coverage-suffix
         tool-name
         button-label
         open-with-label)

(define package-dir
  (let* ([rel-dirs (find-relevant-directories '(multi-file-code-coverage-info-file))])
    (if (> (length rel-dirs) 0)
        (first rel-dirs)
        (current-directory)))
  )

(define info-proc (get-info/full package-dir))

(define (info-look-up name) (if info-proc
                                (info-proc name ( () (symbol->string name)))
                                 ;return something if we fail to find the info file so we dont crash
                                (symbol->string name)))

(define coverage-suffix (info-look-up 'coverage-suffix))
(define tool-name (info-look-up 'tool-name))
(define button-label (info-look-up 'button-label))
(define open-with-label (info-look-up 'open-with-label))
```

tool.rkt

```
#lang racket/base
(require "info-helper.rkt"
         drracket/tool
         drracket/tool-lib
         drracket/private/debug
         drracket/private/rep
         drracket/private/get-extend
         racket/class
         racket/gui/base
         racket/unit
         racket/serialize
         racket/list
         racket/path
         mrlib/switchable-button
         framework)
(provide tool@)

(define tool@
  (unit
    (import drracket:tool^ )
    (export drracket:tool-exports^)
```

```
(define coverage-button-mixin
  (mixin (drracket:unit:frame<%>) ()
    (super-new)
    (inherit get-button-panel
             get-definitions-text
             register-toolbar-button
             get-tabs
             get-current-tab
             get-interactions-text)

    ;Applies code coverage highlighting to all open files. Displays
    ; a dialog with a list of files covered by the currently in focus file
    ; and allows the user to select a file to open and jump to. Coverage
    ; information is first looked for in DrRacket (the program has just been run)
    ; If the program has not been run, or has been modified to invalidate it's
    ; coverage information, look in the compiled directory for coverage info
    ; and display that.
    (define (load-coverage)
      (let* ([current-tab (get-current-tab)]
             [source-file (send (send current-tab get-defs) get-filename)]
             [coverage-file (get-temp-coverage-file source-file)]
             [test-coverage-info-ht (get-test-coverage-info-ht current-tab coverage-file)])
        (when test-coverage-info-ht
          (begin
            (define coverage-report-list (make-coverage-report test-coverage-info-ht coverage-file))

            ;send the coverage info to all files found in the coverage-report
            (map ( (report-item)
                   (let* ([coverage-report-file (string->path (first report-item))]
                          [located-file-tab (locate-file-tab (group:get-the-frame-group) coverage-report-file)])
                     (when located-file-tab
                       (send located-file-tab show-test-coverage-annotations test-coverage-info-ht #f #f #f))
                     ))
                 coverage-report-list)

            (let* ([frame-group (group:get-the-frame-group)]
                   [choice-pair (get-covered-files-from-user
                                 (format "Files covered by ~a" source-file)
                                 (map ( (item)
                                        (format "~a~a (~a%)"
                                                (first item)
                                                (if (second item) "" "*")
                                                (third item)))
                                      coverage-report-list))]
                   [choice-open-with (first choice-pair)]
                   [choice-index-list (last choice-pair)]
                   )
              ;switch to or open a new frame with the selected file and display the uncoverd lines in a new dialog
              (when choice-index-list
                (map ( (choice-index)
                       (let* ([coverage-report-item (list-ref coverage-report-list choice-index)]
                              [coverage-report-file (string->path (first coverage-report-item))]
                              [coverage-report-lines (last coverage-report-item)]
                              [edit-frame (handler:edit-file coverage-report-file)])
                         (when (and choice-open-with (> (length coverage-report-lines) 0))
                           (send (uncovered-lines-dialog coverage-report-file coverage-report-lines) show #t))

                         ;send coverage info to the newly opened file
                         (define located-file-tab (locate-file-tab frame-group coverage-report-file))
                         (when located-file-tab
                           (send located-file-tab show-test-coverage-annotations test-coverage-info-ht #f #f #f))
                         ))
                     choice-index-list))


              )))))

    ;Get the given tab's coverage info, either from drrackets current test-coverage-info (if it has been run) or from
    ; a saved one. If the test coverage does not need to be loaded (the program has recently been run) save it to
    ; coverage-file. If the test coverage needs to be loaded then check if the given tab has
    ; been modified since the coverage was saved and display a warning if it has. If no coverage information can
    ; be found display a warning with suggestions to fix it.
    ;drracket:unit:tab<%> path? -> (or #f test-coverage-info)
    (define (get-test-coverage-info-ht current-tab coverage-file)
      (let* ([source-file (send (send current-tab get-defs) get-filename)]
             [interactions-text (get-interactions-text)]
             [test-coverage-info-drracket (send interactions-text get-test-coverage-info)])
        (if test-coverage-info-drracket ;if DrRacket has some test coverage info
            (begin
              (when coverage-file
                  (save-test-coverage-info test-coverage-info-drracket coverage-file))
              (send interactions-text set-test-coverage-info #f) ;clear out drrackets test-coverage-info-ht so we can use
                                                                 ;our coverage file modified time as a reference for when
                                                                 ;coverage was last run
              test-coverage-info-drracket)

            (if (and coverage-file (file-exists? coverage-file)) ;Maybe we have some saved test coverage info?
                (if (and (not (is-file-still-valid? source-file coverage-file)) ;check if the saved info is up to date
                         (not (out-of-date-coverage-message coverage-file))) ;if its not up to date, ask the user if
                                                                             ;they want to use it anyways
```

```
                    #f
                    (load-test-coverage-info coverage-file))
                (begin ; no test coverage info, tell the user and how they might collect some
                  (no-coverage-information-found-message source-file)
                  #f)
                ))))



    ;Creates the load coverage button and add it to the menu bar in DrRacket
    (define load-button (new switchable-button%
                             (label button-label)
                             (callback ( (button)
                                         (load-coverage)))
                             (parent (get-button-panel))
                             (bitmap code-coverage-bitmap)
                             ))

    (register-toolbar-button load-button)
    (send (get-button-panel) change-children
          ( (l)
            (cons load-button (remq load-button l))))
    ))

;Takes a test-coverage-info-ht and returns a sorted (alphibetical, but with uncovered
;files first) list of file-name coverage-is-valid? %covered uncovered-lines.
;hasheq path -> (list string?  boolean? integer? (list integer?))
(define (make-coverage-report test-coverage-info-ht coverage-file)
  (let* ([file->lines-ht (make-hash)])
    (hash-for-each test-coverage-info-ht
                   ( (key value)
                     (let* ([line (syntax-line key)]
                            [source (format "~a" (syntax-source key))]
                            [covered? (mcar value)]
                            [file->lines-value (hash-ref file->lines-ht
                                                         source
                                                         (list
                                                          (is-file-still-valid? (string->path source) coverage-file)
                                                          0 ;number of lines in the file
                                                          (list) ;list of uncovered lines
                                                          ))])
                       (hash-set! file->lines-ht source
                                  (list (first file->lines-value)
                                        (max line (second file->lines-value))
                                        (if (or covered? (member line (last file->lines-value)))
                                            (last file->lines-value)
                                            (sort (append (last file->lines-value) (list line)) <)))
                                  ))))
    (let* ([test-coverage-info-list (sort (map ( (item) (list (first item)
                                                               (second item)
                                                               (get-percent (length (last item)) (third item))
                                                               (last item)))
                                               (hash->list file->lines-ht))
                                          ( (a b) (< (third a) (third b))))])
      test-coverage-info-list)))

;Determine if coverage-file's coverage info will still be valid for file. If file has been updated/modified
;more recently than coverage-file this indicates that coverage-file's coverage info may not be valid for file
;
(define (is-file-still-valid? file coverage-file)
  (if coverage-file
      (let* ([file-modify-valid? (> (file-or-directory-modify-seconds coverage-file)
                                    (file-or-directory-modify-seconds file))]
             [located-file-frame (send (group:get-the-frame-group) locate-file file)]
             [file-untouched-valid? (if located-file-frame
                                        ;dosnt work for individual tabs inside of a single frame
                                        (not (send (send located-file-frame get-editor) is-modified?))
                                        #t)])
        (and file-modify-valid? file-untouched-valid?)
        )
      #t ;no coverage-file indicating that the source file has not been saved, so they only way we could have coverage info
      ;is if we got drracket's which means it is up to date.
      ))



;Get the name and location of a code coverage file based on the name of a source file or
;#f if the source file has not been saved. Also creates coverage dir if it does not exisit
;path? -> (or path? #f)
(define (get-temp-coverage-file source-file)
  (if source-file
      (let* ([file-base (file-dir-from-path source-file)]
             [file-name (file-name-from-path source-file)]
             [temp-coverage-file-name (path-replace-suffix file-name coverage-suffix)]
             [temp-coverage-dir (build-path file-base "compiled")]
             [temp-coverage-file (build-path temp-coverage-dir temp-coverage-file-name)])
        (when (not (directory-exists? temp-coverage-dir))
          (make-directory temp-coverage-dir))
        temp-coverage-file
        )
      #f))
```

```
;Get a the directory from path
(define (file-dir-from-path path)
  (define-values (file-dir file-name must-be-dir) (split-path path))
  file-dir)

;writes the test-coiverage-info hash table to the given file so that "load-test-coverage-info"
;can reconstruct the hash table
;hasheq path -> void
(define (save-test-coverage-info test-coverage-info coverage-file)
  (with-output-to-file coverage-file
    (lambda () (begin
                 (write (hash-map test-coverage-info
                                  ( (key value)
                                    (cons (list (serialize (syntax-source key))
                                                (syntax-position key)
                                                (syntax-span key)
                                                (syntax-line key))
                                          value))

                                  ))))
      #:mode 'text
      #:exists 'replace
    ))

;Convert the coverage file to a test-coverage-info hash table
;path -> hasheq
(define (load-test-coverage-info coverage-file)
  (make-hasheq (map (lambda (element)
                      (let* ([key (car element)]
                             [value (cdr element)])
                        (cons (datum->syntax #f (void) (list
                                                        (deserialize (car key))
                                                        (cadddr key)
                                                        1
                                                        (cadr key)
                                                        (caddr key)))
                              (mcons (car value) (cdr value)))))
                    (read (open-input-file coverage-file)))))


;Locates the tab with the given file name from a group of frames
(define (locate-file-tab frame-group file)
  (let*  ([located-file-frame (send frame-group locate-file file)]
          [located-file-tab (if located-file-frame
                                (findf ( (t)
                                         (equal?
                                          (send (send t get-defs) get-filename)
                                          file))
                                       (send located-file-frame get-tabs))
                                #f)])
    located-file-tab))

;Do some random math to try and get a decent hight for a listbox based on the number of items in it
(define (get-listbox-min-height num-items)
  (inexact->exact (min 500 (round (sqrt (* 600 num-items))))))

;compute the percent of uncovered lines rounded to a whole percent. Only return 100 the number of
; uncovered lines is exactly 0
(define (get-percent num-uncovered total)
  (if (= num-uncovered 0)
      100
      (min (round (* (- 1 (/ num-uncovered total)) 100)) 99)))

;Display a message warning the user that the code coverage may be out of date
;returns #t if the user clicks "Continue", #f otherwise
(define (out-of-date-coverage-message file)
  (equal?
   (message-box/custom tool-name
                       (string-append (format "The multi-file code coverage information for ~a" file)
                                      " may be out of date, run the program again to update it."
                                      " Do you want to use it anyways?")
                       "Continue" ;button 1
                       "Cancel" ;button 2
                       #f
                       #f
                       (list 'caution 'default=2))
   1))

;Display a message informing the user that no multi-file code coverage info was found and how they might collect some
(define (no-coverage-information-found-message source-file)
  (message-box tool-name
               (format (string-append "No multi-file code coverage information found for ~a. "
                                      "Make sure the program has been run and Syntactic Test Suite Coverage "
                                      "is enabled in Language->Choose Language...->Dynamic Properties.")
                       (if source-file source-file "Untitled"))
               #f
               (list 'ok 'stop))
  )

;Similar to get-choices-from user, but has two open buttons. One with uncovered lines dialog and one without
```

```
;string? (list string?) -> (list boolean? (list integer?))
(define (get-covered-files-from-user message choices)
  (define button-pressed (box 'close))
  (define (button-callback button)
    ( (b e)
      (set-box! button-pressed button)
      (send dialog show #f)))

  (define (enable-open-buttons enable?)
    (send open-button enable enable?)
    (send open-with-button enable enable?)
    (if enable?

        (when (send close-button-border is-shown?)
          (begin
            (send panel delete-child close-button-border)
            (send panel add-child close-button)))

        (when (send close-button is-shown?)
          (begin
            (send panel delete-child close-button)
            (send panel add-child close-button-border))))
  )

  (define dialog (instantiate dialog% (tool-name)))
  (new message% [parent dialog] [label message])
  (define list-box (new list-box%
                        [label ""]
                        [choices choices]
                        [parent dialog]
                        [style '(multiple)]
                        [min-height (get-listbox-min-height (length choices))]
                        [callback ( (c e)
                                    (if (> (length (send list-box get-selections)) 0)
                                        (if (eq? (send e get-event-type) 'list-box-dclick)
                                            ((button-callback 'open) null null)
                                            (enable-open-buttons #t))
                                        (enable-open-buttons #f)))
                                  ]))

  (define panel (new horizontal-panel% [parent dialog]
                     [alignment '(right bottom)]
                     [stretchable-height #f]))


  (define open-button (new button% [parent panel]
                           [label "Open"]
                           [callback (button-callback 'open)]
                           [enabled #f]
                           [style '(border)]))
  (define open-with-button (new button% [parent panel]
                                [label open-with-label]
                                [callback (button-callback 'open-with)]
                                [enabled #f]))
  (define close-button-border (new button% [parent panel]
                                   [label "Close"]
                                   [style '(border)]
                                   [callback (button-callback 'close)]))
  (define close-button (new button% [parent panel]
                            [label "Close"]
                            [style '(deleted)]
                            [callback (button-callback 'close)]))

  (send dialog show #t)
  (case (unbox button-pressed)
    ['open (list #f (send list-box get-selections))]
    ['open-with (list #t (send list-box get-selections))]
    [else (list #f #f)])
  )

; Dialog that displays the uncovered lines. Not a message box so the user can still interact
; with DrRacket without having to close it.
(define (uncovered-lines-dialog file lines)
  (let* ([dialog (instantiate frame% (tool-name))])
    (new message%
         [parent dialog]
         [label "Lines containing uncovered code in"])
    (new message%
         [parent dialog]
         [label (format "~a:" file)])
    (define text-field (new text-field%
         [label ""]
         [parent dialog]
         [style (list 'multiple)]))
    (send text-field set-value (foldl ( (item text)
                              (string-append text
                                             (if (equal? text "") "" ", ")
                                             (number->string item)))
                            "" lines))
    (send (send text-field get-editor) lock #t)
    (define panel (new horizontal-panel%
                       [parent dialog]
                       [stretchable-height #f]
                       [alignment '(right bottom)]))

    (new button%
         [parent panel]
         [label "Close"]
         [style '(border)]
         [callback ( (b e) (send dialog show #f))])
    dialog))

; Graphic for the code coverage button
(define code-coverage-bitmap
  (let* ((bmp (make-bitmap 16 16))
         (bdc (make-object bitmap-dc% bmp)))
    (send bdc erase)
    (send bdc set-smoothing 'smoothed)
    (send bdc set-pen "black" 1 'transparent)
    (send bdc set-brush "forest green" 'solid)
    (send bdc draw-rectangle 2 5 12 9)
    (send bdc set-brush "maroon" 'solid)
    (send bdc draw-rectangle 11 5 14 9)
    (send bdc set-bitmap #f)
    bmp))

(define (phase1) (void))
(define (phase2) (void))

(drracket:get/extend:extend-unit-frame coverage-button-mixin)))
```

B PLaneT Documentation

Multi-File Code Coverage Tool

May 20, 2011

The Multi-File Code Coverage Tool allows coverage information gathered from a single program evaluation to be displayed on multiple source files in multiple DrRacket windows.

1 Installing the Tool

Multi-File Code Coverage is a Planet package, however it only adds a tool rather than providing functions. To install evaluate the following program:

```
#lang racket
(require (planet jowalsh/code-coverage))
```

... and then restart DrRacket.

2 Using the Tool

First ensure that you have "Syntactic Test Suite Coverage" enabled in the "Language->Choose Language..." dialog. Then run the program you wish to collect multi-file coverage information for. Finally, click the "Multi-File Coverage" button. A new dialog will apear with the list of file covered by the program you just ran.

You may then select one, or more, of the covered files to open and switch focus to. Additinaly, by clicking the "Open With Uncovered Lines Dialog", instead of just "Open", each selected file will spawn an additional dialog with a list of lines containing unevaluated expressions.

2.1 Line Coverage Percent

The percent in parentheses, next to each file, is the percent of covered lines in that file. So 100% indicates that every line in that file is covered. Blank lines and comments are also counted as covered lines.

2.2 Invalid Coverage Information

An asterisk may appear next to files in the covered files dialog. This indicates that the file has been modified since multi-file coverage information was last collected, which may have invalidated its coverage info. To ensure that all coverage information is valid run the program you are collecting multi-file code coverage for again.

2.3 Saving Coverage Information

Whenever the "Multi-File Coverage" button is clicked the multi-file code coverage information is saved to a file. This file is named <file name>..rktcov and saved in the "compiled" directory next to the source file.

2.4 Why doesn't my source file show up in the covered files dialog?

Only un-compiled files will appear in the covered files dialog. To ensure that all your covered files appear delete any "compiled" directories next to your source files. Then run your program again and click the "Multi-File Coverage" button.