GNU Emacs package: rcd-dashboard.el — Tool to build Emacs dashboards

Table of Contents

The RCD Dashboard package generates temporary Emacs buffer with options and links to invoke various Emacs functions.

About Computer Dashboards

From reference: https://en.wikipedia.org/wiki/Dashboard_(business)

In business computer information systems, a dashboard is a type of graphical user interface which often provides at-a-glance views of key performance indicators (KPIs) relevant to a particular objective or business process. In other usage, "dashboard" is another name for "progress report" or "report" and considered a form of data visualization. In providing this overview, business owners can save time and improve their decision making by utilizing dashboards.

Customization of RCD Dashboard

RCD Dashboard package is comptaible with Emacs 29 version.

If you wish the RCD Dashboard buffer to appear in full window, customize the variable `rcd-dashboard-full-window` which is set to TRUE by defaul.

Required GNU Emacs package: rcd-utilities.el:
https://gnu.support/gnu-emacs/packages/rcd-utilities-el.html

To install this package:

```
{M-x package-install-file RET rcd-dashboard.el RET}
```

or download the file from: https://gnu.support/files/emacs/packages/rcd-dashboard.el

and install it with:

```
{M-x package-install-file RET rcd-dashboard.el RET}
```

This file `rcd-dashboard.el` will be updated.

How to use RCD Dashboard for Emacs

First you define functions you wish to appear in the dashboard:

```
(defun rcd-org-agenda-header ()
  "RCD Org Agenda Dashboard header."
  (rcd-dashboard-heading
   (format "□ RCD Org Agenda □ Dashboard □%s"
       (cond (user-full-name (format " for %s" user-full-name))
         (t "")))))

(defun rcd-org-agenda-functions ()
  "Ordinary Org Agenda functions for RCD Org Agenda."
  (rcd-dashboard-button "Agenda for current week or day" (lambda (_) (org-agenda-
list)))
  (rcd-dashboard-button "List of all TODO entries" (lambda (_) (org-todo-list)))
  (rcd-dashboard-button "Match a TAGS/PROP/TODO query" (lambda (_) (org-tags-view)))
  (rcd-dashboard-button "Search for keywords" (lambda (_) (org-call-with-arg 'org-
search-view '(4))))
  (rcd-dashboard-button "Search for keywords within TODO entries" (lambda (_)
                                (org-call-with-arg 'org-search-view '(4))))
  (rcd-dashboard-button "Export agenda views" (lambda (_) (call-interactively 'org-
store-agenda-views)))
  (rcd-dashboard-button "Multi-occur" (lambda (_) (call-interactively #'org-occur-in-
agenda-files)))
  (rcd-dashboard-button "Find :FLAGGED: entries" (lambda (_) (org-tags-view nil
"+FLAGGED")))
  (insert "\n"))
```

And once functions are defined, you invoke them by using `rcd-dashboard`:

```
(defun rcd-org-agenda-dashboard ()
  "RCD Org Agenda Dashboard."
  (interactive)
  (let ((rcd-dashboard-buffer-name "RCD Org Agenda Dashboard"))
    (rcd-dashboard
     '(rcd-org-agenda-header
       rcd-org-agenda-functions
       read-only-mode)
     "rcd-org-agenda")))
```

With the following result being non-blocking Org mode preview that allows free mouse and keyboard usage:

[Screenshot 2022 12 25 18 09 08 704834734] | *https://gnu.support/files/emacs/packages/rcd-*

dashboard/Screenshot-2022-12-25-18-09-08-704834734.jpg

You may get the PDF file of this page.

Source of rcd-dashboard.el

```
;;; rcd-dashboard.el --- RCD Dashboard  -*- lexical-binding: t; -*-

;; Copyright (C) 2022 by Jean Louis

;; Author: Jean Louis <bugs@gnu.support>
;; Version: 0.01
;; Package-Requires: (org rcd-utilities)
;; Keywords: convenience
;; URL:

;; This file is not part of GNU Emacs.

;; This program 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.
;;
;; 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
;; General Public License for more details.
;;
;; You should have received a copy of the GNU General Public License
;; along with this program. If not, see <http://www.gnu.org/licenses/>.

;;; Commentary:
;;
;; The RCD Dashboard package generates temporary Emacs buffer with
;; options and links to invoke various Emacs functions.
;;
;; From reference: https://en.wikipedia.org/wiki/Dashboard_(business)
;;
;; In business computer information systems, a dashboard is a type of
;; graphical user interface which often provides at-a-glance views of
;; key performance indicators (KPIs) relevant to a particular
;; objective or business process. In other usage, "dashboard" is
;; another name for "progress report" or "report" and considered a
;; form of data visualization. In providing this overview, business
;; owners can save time and improve their decision making by utilizing
;; dashboards.
;;
;; RCD Dashboard provides very fundamental functions for dashboard
;; generation.
```

```
;;
;; Required GNU Emacs package: rcd-utilities.el:
;; https://gnu.support/gnu-emacs/packages/rcd-utilities-el.html
;;
;; If you wish the RCD Dashboard buffer to appear in full window,
;; customize the variable `rcd-dashboard-full-window' which is set to
;; TRUE by defaul.

;;; Change Log:

;;; Code:

;;; Required Libraries

(require 'org)
(require 'rcd-utilities)

;;; Custom variables

(defcustom rcd-dashboard-full-window t
  "When set RCD Dashboard will appear in full window."
  :group 'rcd
  :type 'boolean)

;;; Variables

(defvar-local rcd-dashboard-buffer-name "RCD Dashboard"
  "The RCD Dashboard buffer name")
(put 'rcd-dashboard-name 'permanent-local t)

(defvar-local rcd-dashboard-invoking-buffer nil
  "Name of the RCD Dashboard invoking buffer.")

;; There may be many invoking buffers, so we make the variable
;; `rcd-dashboard-invoking-buffer' permanently local.
(put 'rcd-dashboard-invoking-buffer 'permanent-local t)

;; (defvar-local rcd-dashboard-button-character = nil
;;   "Name of the RCD Dashboard invoking buffer.")

(defvar rcd-dashboard-symbol nil
  "Used in RCD Dashboard text properties.")

(defvar-local rcd-dashboard-last-position 0
  "Remember the last position in RCD Dashboard.")

;; There may be many various dashboards, so we make the variable
;; `rcd-dashboard-last-position' permanently local, so that cursor
;; position may be remembered for each dashboard buffer.
(put 'rcd-dashboard-last-position 'permanent-local t)
```

```
;;; RCD Dashboard Mode

(defun rcd-dashboard-mode-name (dashboard-mode-name)
  "Return RCD Dashboard mode name for new dashboard mode.

It is necessary to create new derived mode for buffer local
variables to work properly."
  (concat "rcd-dashboard-" dashboard-mode-name "-mode"))

(defun rcd-dashboard-derived-mode (dashboard-mode-name &optional name docstring)
  "Define new derived mode named `rcd-dashboard-DASHBOARD-MODE-NAME-mode'.

NAME is optional or otherwise generated automatically.
DOCSTRING is optional or otherwise generated automatically."
  (let* ((my-mode-name-map (intern (concat dashboard-mode-name "-map")))
     (my-mode (intern dashboard-mode-name)))
    (eval
     `(define-derived-mode ,my-mode org-mode
    ,(or name (format "RCD Dashboard Mode (%s)" dashboard-mode-name))
    ,(or docstring (format "RCD Dashboard (%s) is derived from Org mode." dashboard-
mode-name))))
    (keymap-set (symbol-value my-mode-name-map) "C-x k" #'rcd-dashboard-kill-buffer)
    (keymap-set (symbol-value my-mode-name-map) "q" #'rcd-dashboard-quit-window)
    (keymap-set (symbol-value my-mode-name-map) "Q" #'rcd-dashboard-kill-buffer)
    (keymap-set (symbol-value my-mode-name-map) "C-c C-c" #'rcd-dashboard--toggle-
variable)))

;;; RCD Dashboard Org Heading Button

(defun rcd-dashboard-button (name function &optional heading-level how-many)
  "Insert Org heading NAME with RCD button FUNCTION.

HEADING-LEVEL is optional integer by default 2.
HOW-MANY is optional function to disply superscript count."
  (let* ((heading-level (or heading-level 2))
     (heading-prefix (concat "\n" (make-string heading-level ?*) " ")))
  (insert heading-prefix)
  (rcd-button-insert name function how-many)))

;;; RCD Dashboard symbols and variables toggling

(defun rcd-dashboard--toggle-variable ()
  "Toggle symbol value found in text property RCD-DASHBOARD-SYMBOL."
  (interactive)
  (rcd-properties--toggle-variable-and-rcd-check 'rcd-dashboard-symbol "[X]" "[X]"))

(defun rcd-dashboard-button-check-item-toggle (check-item symbol key check-in check-
out rcd-dashboard-mode-name
                               &optional tip prefix symbol-name)
  "Insert button CHECK-ITEM and enable toggling of SYMBOL value found inside.
```

```
SYMBOL represents symbol of global variable.

Variables are toggled to be TRUE or FALSE.

CHECK-IN may be \"[ ]\", and CHECK-OUT \"[X]\" for Org mode. It
may be anything else.

Variable is expected to be global to function for it to work.

TIP is text that appears after the button.

PREFIX when available is inserted before the CHECK-IN."
  (when (boundp symbol)
    (let* ((my-line (line-number-at-pos (point)))
       (function (lambda (&rest _)
               (interactive)
               (rcd-variable-toggle-global symbol)
               (save-excursion
             (read-only-mode 0)
             (goto-char (point-min))
             (forward-line (1- my-line))
             (rcd-check check-in check-out)
             (read-only-mode 1)))))
      (keymap-set rcd-dashboard-mode-name key function)
      (when prefix (insert prefix))
      (insert (cond ((symbol-value symbol) check-out)
            (t check-in)))
      (insert " ")
      (rcd-button-insert check-item
             (lambda (_)
               (interactive)
               (rcd-variable-toggle-global symbol)
               (read-only-mode 0)
               (rcd-check check-in check-out)
               (read-only-mode 1))
             nil (or symbol-name "rcd-symbol") symbol)
      (when tip (insert " " tip)))))

;;; RCD Dashboard main functions

(defun rcd-dashboard-pop-to-buffer (invoking-buffer dashboard-mode-name)
  "Pop up RCD Dashboard from INVOKING-BUFFER."
  (let ((buffer (get-buffer-create rcd-dashboard-buffer-name)))
    (set-buffer buffer)
    (funcall (intern dashboard-mode-name))
    (read-only-mode 0)
    (setq rcd-dashboard-invoking-buffer invoking-buffer)
    (erase-buffer)
    (cond (rcd-dashboard-full-window
       (switch-to-buffer buffer)
       (delete-other-windows))
```

```
      (t (pop-to-buffer buffer)))))

(defun rcd-dashboard-heading (heading &optional description)
  "Insert RCD Dashboard HEADING string.

DESCRIPTION is optional text to appear under the HEADING."
  (read-only-mode 0)
  (insert "\n* " heading "\n")
  (when description (insert "\n" description "\n")))

;;; Quit or kill RCD Dashboard and return to invoking buffer.

(defun rcd-dashboard-quit-window (&optional _)
  "Quit window, and switch to invoking buffer."
  (interactive)
  (let ((invoking-buffer rcd-dashboard-invoking-buffer))
    (setq rcd-dashboard-last-position (point))
    (quit-window)
    (when (buffer-live-p invoking-buffer)
      (switch-to-buffer invoking-buffer))))

(defun rcd-dashboard-kill-buffer (&optional _)
  "Kill buffer, and switch to invoking buffer."
  (interactive)
  (let ((invoking-buffer rcd-dashboard-invoking-buffer))
  (setq rcd-dashboard-last-position (point))
  (kill-current-buffer)
    (when (buffer-live-p invoking-buffer)
      (switch-to-buffer invoking-buffer))))

;;; RCD Dashboard

(defun rcd-dashboard (list-of-functions &optional dashboard-name)
  "Start RCD Dashboard and invoke LIST-OF-FUNCTIONS."
  (let* ((name (or dashboard-name "main"))
     (my-mode-name (rcd-dashboard-mode-name name)))
    (rcd-dashboard-derived-mode my-mode-name)
    (let ((invoking-buffer (current-buffer)))
      (rcd-dashboard-pop-to-buffer invoking-buffer my-mode-name)
      (mapc #'funcall list-of-functions)
      (goto-char rcd-dashboard-last-position))))

(provide 'rcd-dashboard)

;;; rcd-dashboard.el ends here
```

Source of this page

It may be interesting to see how source of this page looks like.

```
:source-highlighter: rouge
:icons: font
:toc:

The RCD Dashboard package generates temporary Emacs buffer with
options and links to invoke various Emacs functions.

== About Computer Dashboards

From reference: <https://en.wikipedia.org/wiki/Dashboard_(business)>

In business computer information systems, a dashboard is a type of
graphical user interface which often provides at-a-glance views of
key performance indicators (KPIs) relevant to a particular
objective or business process. In other usage, "dashboard" is
another name for "progress report" or "report" and considered a
form of data visualization. In providing this overview, business
owners can save time and improve their decision making by utilizing
dashboards.

== Customization of RCD Dashboard

CAUTION: RCD Dashboard package is comptaible with Emacs 29 version.

If you wish the RCD Dashboard buffer to appear in full window,
customize the variable `rcd-dashboard-full-window` which is set to
TRUE by defaul.

Required GNU Emacs package: rcd-utilities.el: +
https://gnu.support/gnu-emacs/packages/rcd-utilities-el.html

To install this package:

```
{M-x package-install-file RET rcd-dashboard.el RET}
```

or download the file from: https://gnu.support/files/emacs/packages/rcd-dashboard.el

and install it with:

```
{M-x package-install-file RET rcd-dashboard.el RET}
```

This file `rcd-dashboard.el` will be updated.

== How to use RCD Dashboard for Emacs

First you define functions you wish to appear in the dashboard:
```

```
[source,lisp]
 ----
(defun rcd-org-agenda-header ()
  "RCD Org Agenda Dashboard header."
  (rcd-dashboard-heading
   (format "□ RCD Org Agenda □ Dashboard □%s"
       (cond (user-full-name (format " for %s" user-full-name))
         (t "")))))

(defun rcd-org-agenda-functions ()
  "Ordinary Org Agenda functions for RCD Org Agenda."
  (rcd-dashboard-button "Agenda for current week or day" (lambda (_) (org-agenda-
list)))
  (rcd-dashboard-button "List of all TODO entries" (lambda (_) (org-todo-list)))
  (rcd-dashboard-button "Match a TAGS/PROP/TODO query" (lambda (_) (org-tags-view)))
  (rcd-dashboard-button "Search for keywords" (lambda (_) (org-call-with-arg 'org-
search-view '(4))))
  (rcd-dashboard-button "Search for keywords within TODO entries" (lambda (_)
                                  (org-call-with-arg 'org-search-view '(4))))
  (rcd-dashboard-button "Export agenda views" (lambda (_) (call-interactively 'org-
store-agenda-views)))
  (rcd-dashboard-button "Multi-occur" (lambda (_) (call-interactively #'org-occur-in-
agenda-files)))
  (rcd-dashboard-button "Find :FLAGGED: entries" (lambda (_) (org-tags-view nil
"+FLAGGED")))
  (insert "\n"))
 ----

And once functions are defined, you invoke them by using
`rcd-dashboard`:

[source,lisp]
 ----
(defun rcd-org-agenda-dashboard ()
  "RCD Org Agenda Dashboard."
  (interactive)
  (let ((rcd-dashboard-buffer-name "RCD Org Agenda Dashboard"))
    (rcd-dashboard
     '(rcd-org-agenda-header
       rcd-org-agenda-functions
       read-only-mode)
     "rcd-org-agenda")))
 ----

With the following result being non-blocking Org mode preview that
allows free mouse and keyboard usage:

image::https://gnu.support/files/emacs/packages/rcd-dashboard/Screenshot-2022-12-25-
18-09-08-704834734.jpg[width=800]
```

```
NOTE: You may get the https://gnu.support/files/emacs/packages/rcd-dashboard/GNU-
Emacs-package-rcd-dashboard-el-Tool-to-build-Emacs-dashboards-76668.pdf[PDF file of
this page].

== Source of rcd-dashboard.el

```lisp
 (file-to-string "/home/admin/Programming/emacs-lisp/rcd-dashboard.el") 
```

== Source of this page

It may be interesting to see how source of this page looks like.

 (string-replace "\n----" "\n ----" (hyperscope-text 76668))

```