CS 596 Functional Programming and Design
Fall Semester, 2014
Doc 18 Seesaw GUI
Nov 6, 2014

Seesaw

Library/DSL for constructing GUI's in Clojure
Built on Java Swing

Resources

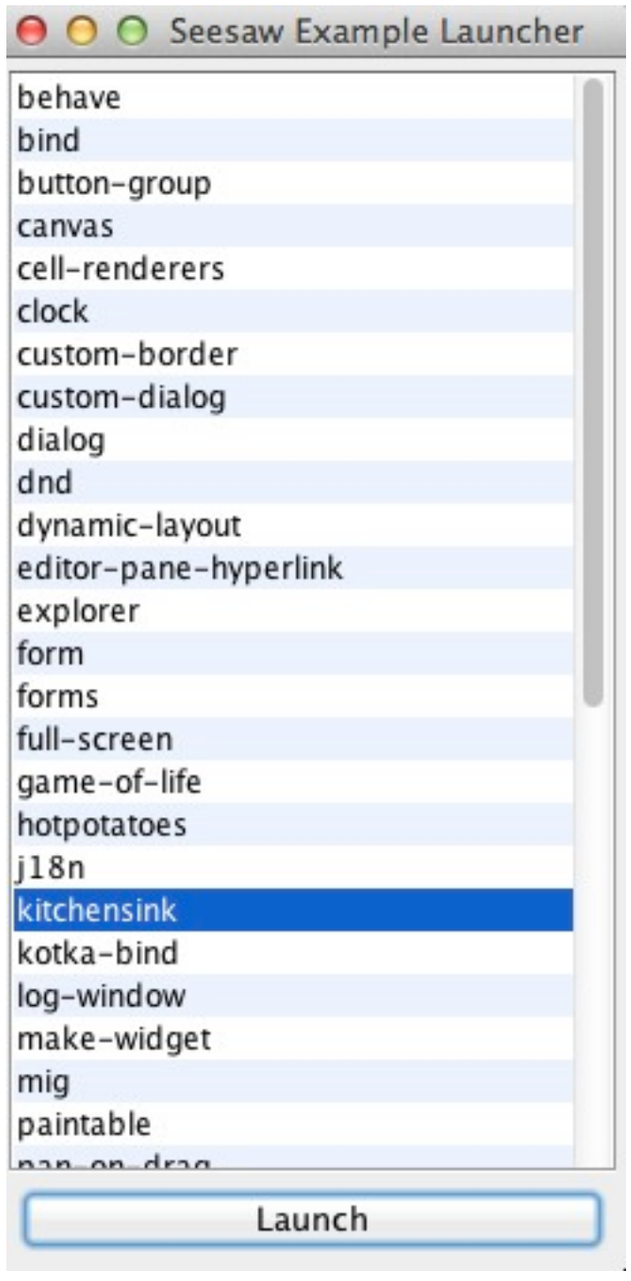
Seesaw github
https://github.com/daveray/seesaw

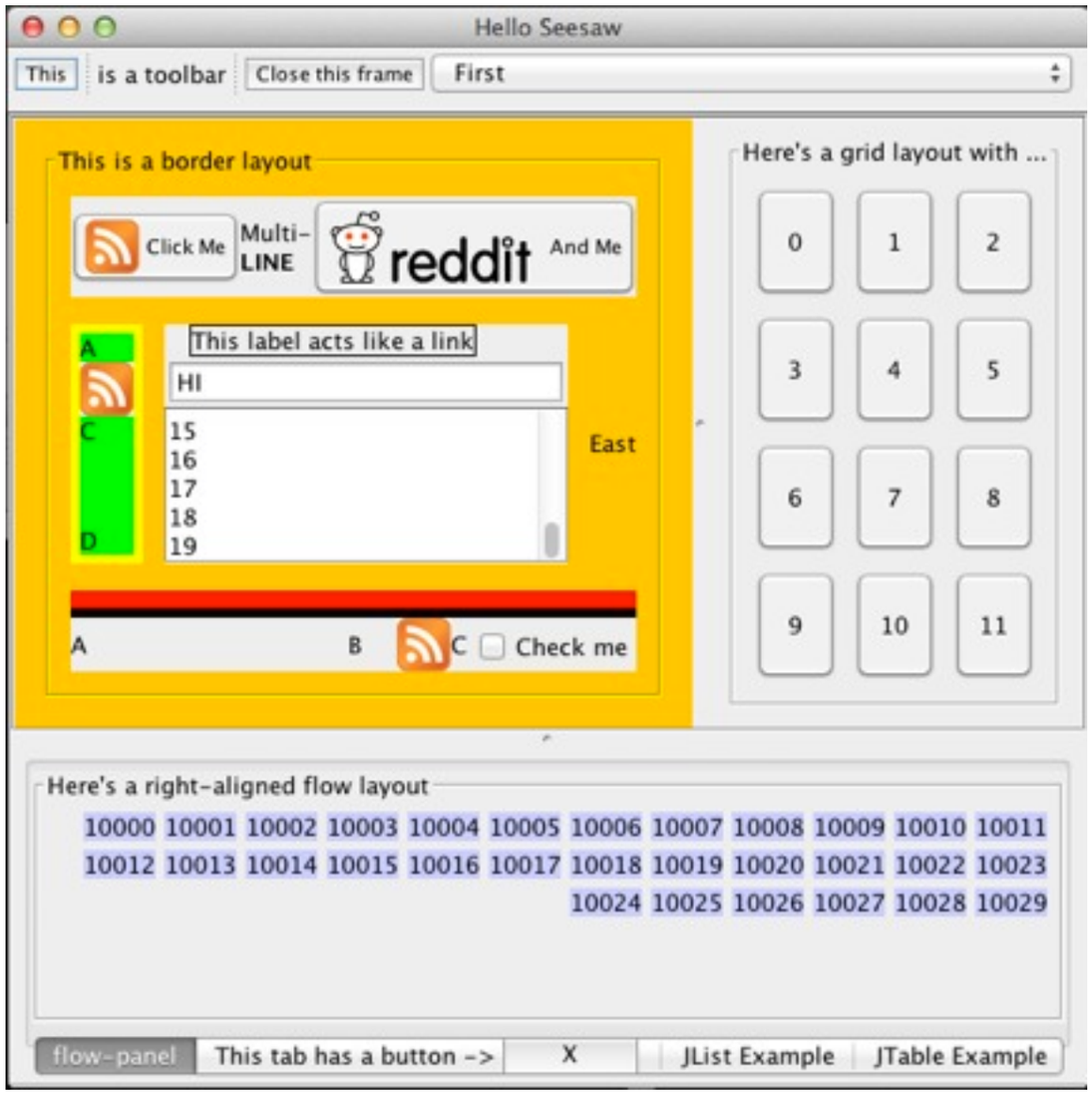
Seesaw Wiki/Overview
https://github.com/daveray/seesaw/wiki

Seesaw Tutorial
https://gist.github.com/daveray/1441520

Seesaw Examples

Download Seeasaw source from github





Using Seesaw

Create lein project

lein new app Your-project-names

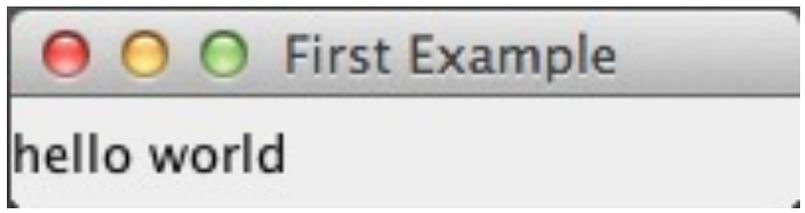
project.clj

```
:dependencies [[org.clojure/clojure "1.6.0"]
               [seesaw "1.4.4"]]
```

```
(ns your-project-name.namespace
  (:gen-class)
  (:require [seesaw.core :as seesaw])
```

First Example

```
(def window
     (seesaw/frame
          :title "First Example"
          :content "hello world"
          :width 200
          :height 50))
(seesaw/show! window)
```



Building a Runable App

Project name gui
File name: core.clj

```
(ns gui.core
  (:gen-class)
  (:require [seesaw.core :as seesaw]))

(def window (seesaw/frame
             :title "First Example"
             :content "hello world"
             :width 200
             :height 50))

(defn -main
  [& args]
  (seesaw/show! window))
```

In Project Directory

lein uberjar

Generates

target/uberjar/gui-0.1.0-Snapshot-standalone.jar

The stand alone jar when executed runs the project main

Native OS features

```
(seesaw/native!)

(def window (seesaw/frame
              :title "First Example"
              :content "hello world"
              :width 200
              :height 50))

(seesaw/show! window)
```

What are the Options?

```
(def window
        (seesaw/frame
            :title "First Example"
            :content "hello world"
            :width 200
            :height 50))
```

seesaw.dev/show-options

```
(ns gui.core
  (:gen-class)
  (:require [seesaw.core :as seesaw]
            [seesaw.dev :as dev])


(def window (seesaw/frame
             :title "First Example"
             :content "hello world"))

(dev/show-options window)
```

Option	
:class	
:content	
:icon	
:icons	
:id	
:listen	
:menubar	
:minimum-size	
:on-close	
:resizable	
:resource	
:size	
:title	
:transfer-handler	
:undecorated?	
:visible?	

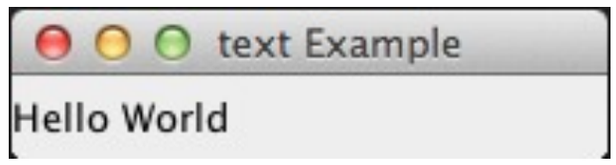
show!, move!, hide!

```
(move! target :to [x, y])
(move! target :by [dx, dy])
```

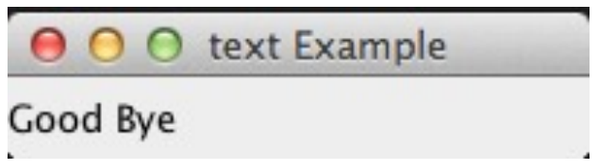
```
(def window (seesaw/frame
             :title "First Example"
             :content "hello world"))

(seesaw/show! window)
(seesaw/move! window :by [100 300])
```

Making Changes

```
(def window (seesaw/frame
             :title "text Example"
             :content "Hello World"
             :width 200
             :height 50))
(seesaw/show! window)
```




```
(seesaw/config! window :content "Good Bye")
```




```
(seesaw/dispose! window)
```

alert

Modal

(seesaw/alert "Hello World")
    Centered in middle of screen

(seesaw/alert window "Hello World")
    Centered in window

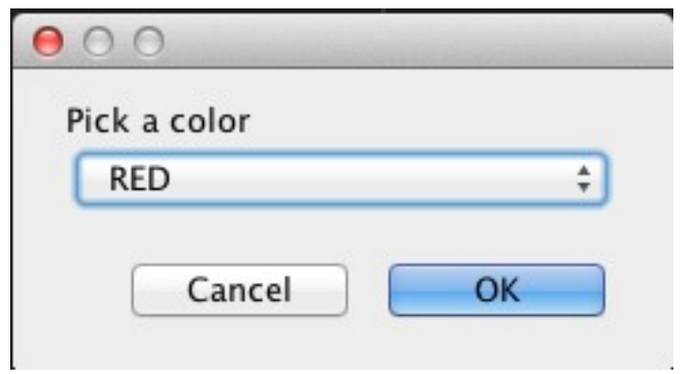
input

Modal

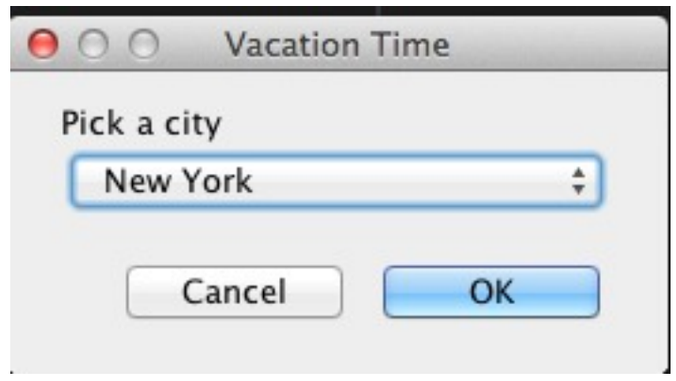
Returns user input or nil if canceled

```
(seesaw/input "Bang the keyboard like a monkey")
```


```
(seesaw/input "Pick a color"
    :choices ["RED" "YELLO" "GREEN"])
```




```
(seesaw/input "Pick a city"
   :choices [{ :name "New York"  :population 8000000 }
             { :name "Ann Arbor" :population 100000 }
             { :name "Twin Peaks" :population 5201 }]
   :to-string :name
   :title "Vacation Time")
```



To Make Examples Simpler

```
(defn display
  [content]
  (let [window (seesaw/frame :title "Example")]
    (-> window
        (seesaw/config! :content content)
        (seesaw/pack!)
        (seesaw/show!))))
```

```
(display "Hi")
```

Button

```
(def button
  (seesaw/button
   :text "Next"
   :listen [:action (fn [event](seesaw/alert "Next!" ))]))

(display button)
```

Thursday, November 6, 14

Alert appears in the middle of the screen

Button

```
(def button
  (seesaw/button
   :text "Next"
   :listen [:action (fn [event] (seesaw/alert event "Next!" ))]))

(display button)
```

Thursday, November 6, 14

Alert appears in the middle of the window

Mouse Events

```
(def button
  (seesaw/button
   :text "Next"
   :listen [:action (fn [event](seesaw/alert event "Next!" ))
            :mouse-entered #(seesaw/config! % :foreground :blue)
            :mouse-exited #(seesaw/config! % :foreground :red)]))
```

Finding out Possible Events

(seesaw.dev/show-events (seesaw/button))

In Console

```
:action [java.awt.event.ActionListener]
  :action-performed
:change [javax.swing.event.ChangeListener]
  :state-changed
:component [java.awt.event.ComponentListener]
  :component-hidden
  :component-moved
  :component-resized
  :component-shown
:focus [java.awt.event.FocusListener]
  :focus-gained
  :focus-lost
:item [java.awt.event.ItemListener]
  :item-state-changed
:key [java.awt.event.KeyListener]
  :key-pressed
  :key-released
  :key-typed
:mouse [java.awt.event.MouseListener]
  :mouse-clicked
```

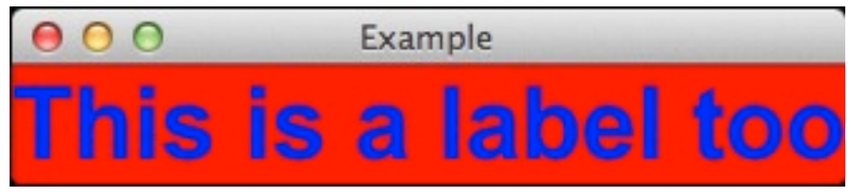
Adding Listeners Separate

```
(def button
  (seesaw/button
   :text "Next"))

(seesaw/listen button
          :action (fn [event](seesaw/alert event "Next!" ))
          :mouse-entered #(seesaw/config! % :foreground :blue)
          :mouse-exited #(seesaw/config! % :foreground :red))
```

Labels

```
(display "This is a label")
```


```
(def label (seesaw/label
                    :text "This is a label too"
                    :background :red
                    :foreground "#00f"
                    :font "ARIAL-BOLD-40"))
(display label)
```



Colors

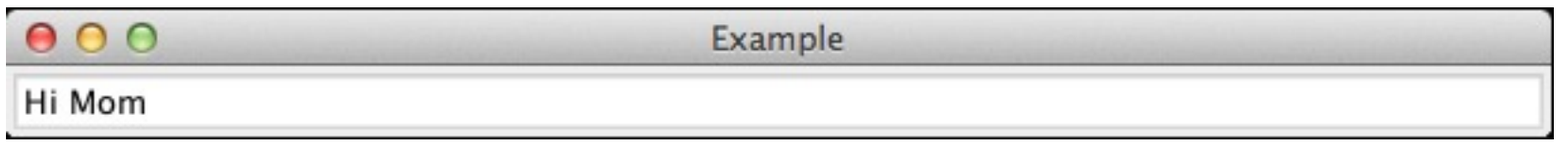
```
:foreground java.awt.Color/BLACK      (a raw color object)
:foreground (color 255 255 224)       (RGB bytes)
:foreground (color 255 255 224 128)   (RGBA bytes)
:foreground "#FFEEDD"                 (hex color string or keyword)
:foreground "#FED"                    ("short" CSS-style hex color string or keyword)
:foreground "aliceblue"               (CSS-style named color string or keyword)
:foreground (color "#FFEEDD" 128)     (hex color string (or name) + alpha)
```

Fonts

```
:font "ARIAL-BOLD-18"                                (Swing-style font spec string)
:font {:name "ARIAL" :style :bold :size 18}          (using a properties hash)
:font (font :name "ARIAL" :style :bold :size 18)     (using properties with font function)
```

Text Fields

```
(def textfield
  (seesaw/text
        :text "Hi Mom"
        :editable? false
        :columns 50))

(display textfield)
```



Getting text from Text or Other Widget

(def textfield (seesaw/text :text "Hi Mom"))

(seesaw/text textfield) returns “Hi Mom”

Given a widget, document, or event seesaw/text returns the text of the argument

Changing the Text

```
(def textfield (seesaw/text :text "Hi Mom"))
```

```
(seesaw/config! textfield :text "This is too long")
```

```
(seesaw/text! textfield "This is better")
```

Getting text/value from Widgets

(def textfield (seesaw/text :text "Hi Mom"))

(seesaw/**value** textfield) returns “Hi Mom”

(seesaw/value target)

Target is a widget

Return natural value of widget

Usually the text or current selection of widget

Target is container

Map of all children widget values

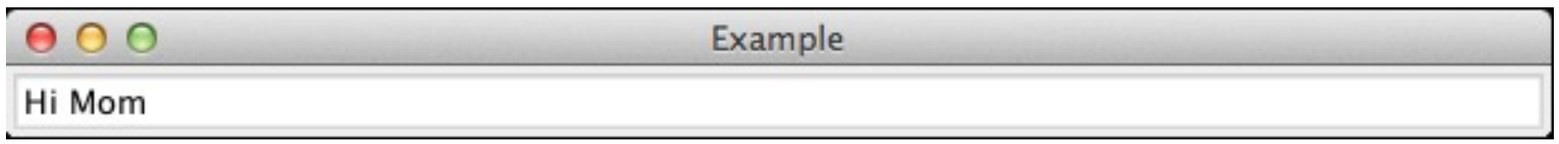
Yes you can set the value

```
(def textfield (seesaw/text :text "Hi Mom"))
```

```
(seesaw/value! textfield "Hi Dad")
```

Text Fields

```
(def textfield
  (seesaw/text
        :text "Hi Mom"
        :editable? true
        :columns 10
        :listen [:document #(println (seesaw/text %))]))

(display textfield)
```



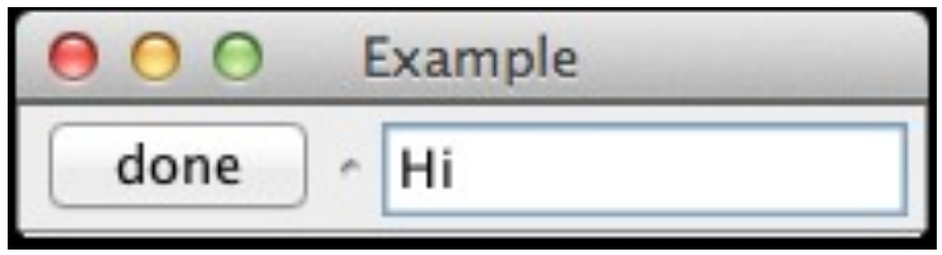
Every time the text changes the :document action is triggered

(seesaw/text event) returns the text of the event source

Containers -Showing More that One Widget

top-bottom-split
left-right-split
border-panel
flow-panel
grid-panel
horizontal-panel
tabbed-panel
vertical-panel

Left-Right-Split




```
(defn two-widgets
  []
  (let [message (seesaw/text :text "hi" :columns 10)
        done (seesaw/button
              :text "done"
              :listen [:action (fn [e] (println (seesaw/text message)))])]
    (seesaw/left-right-split done message)))

(display (two-widgets))
```

When button is pressed current text of the text area is printed out

Finding Widgets in Window

Handlers do not always have direct access to a widget

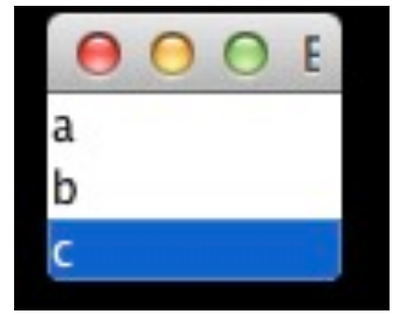
Give widget an id

(seesaw/to-root event) returns the root widget

(seesaw/select root [:#the-id-for-widget]) returns widget

```
(defn done
  [event]
  (let [root (seesaw/to-root event)
        textfield (seesaw/select root [:#foo])
        current-text (seesaw/text textfield)]
    (seesaw/text! textfield (clojure.string/capitalize current-text))))


(defn two-widgets
  []
  (let [message (seesaw/text :text "hi" :columns 10 :id :foo)
        done (seesaw/button
              :text "done"
              :listen [:action done])]
    (seesaw/left-right-split done message)))
```

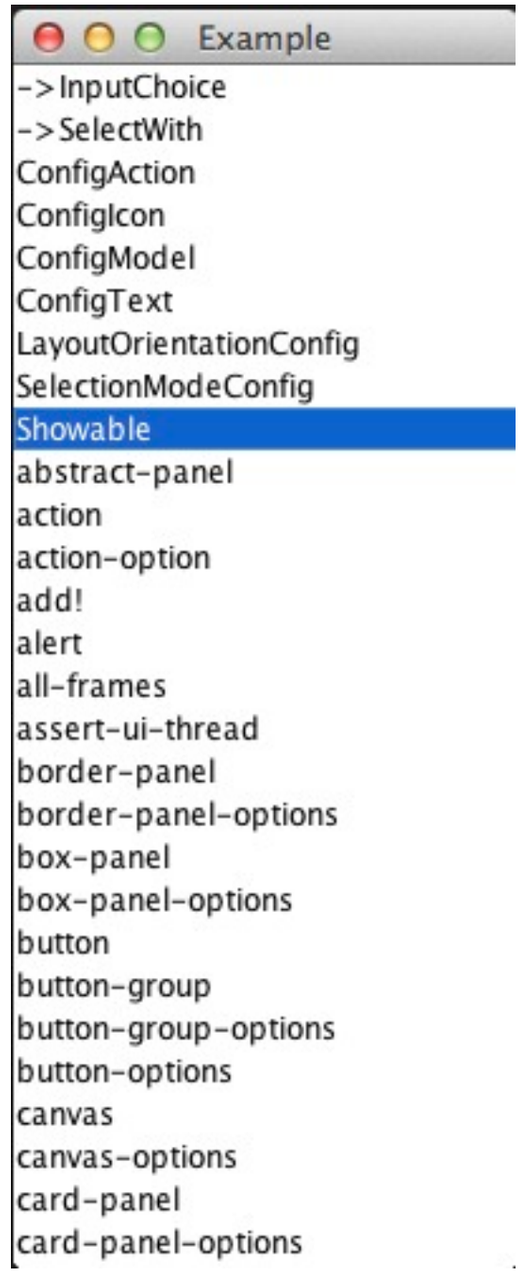
Lists




```
(def small-list (seesaw/listbox :model ["a" "b" "c"]))

(seesaw/listen small-list
               :selection (fn [event] (println "You selected " (seesaw/selection event))))

(display small-list)
```

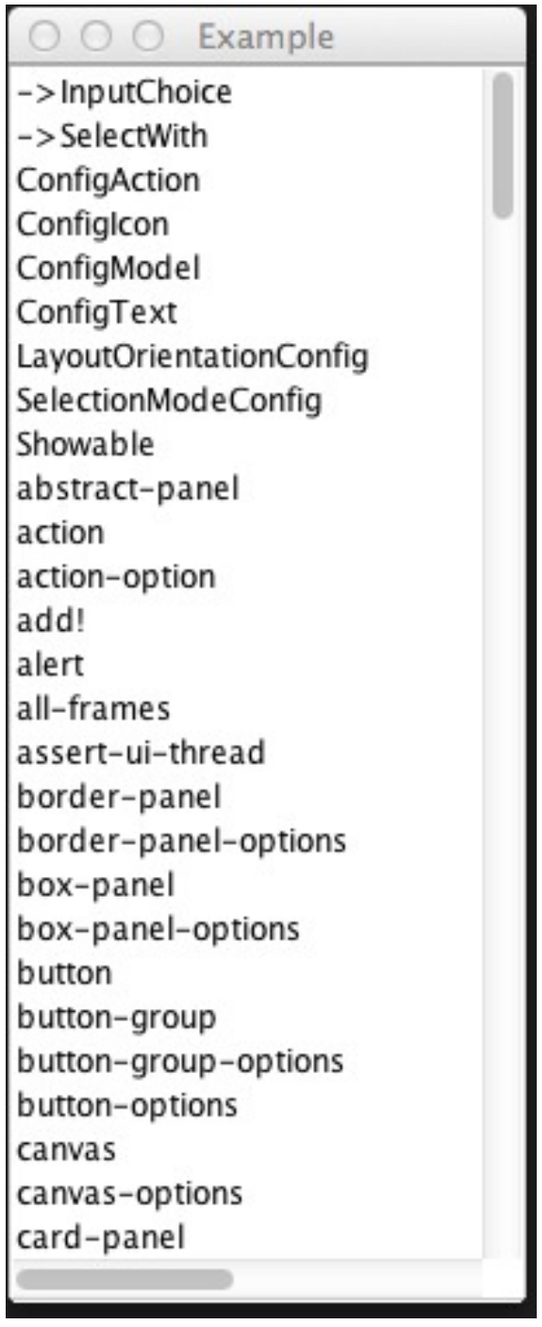
What Happens when list is too long!

```
(defn display
  [content width height]
  (let [window (seesaw/frame :title "Example"
                             :content content
                             :width width
                             :height height)]
       (seesaw/show! window)))

(def small-list (seesaw/listbox
       :model (-> 'seesaw.core ns-publics keys sort)))

(display small-list 150 100)
```



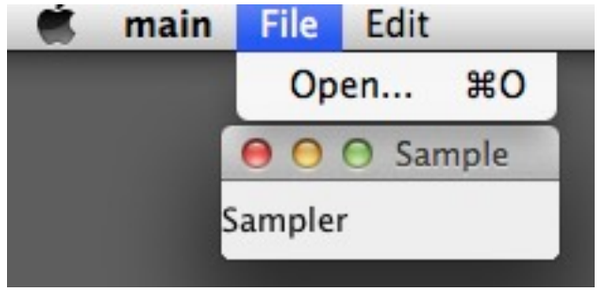
Scrolling - seesaw/scrollable

```
(def small-list (seesaw/listbox
                 :model (-> 'seesaw.core ns-publics keys sort)))

(display (seesaw/scrollable small-list) 200 500)
```



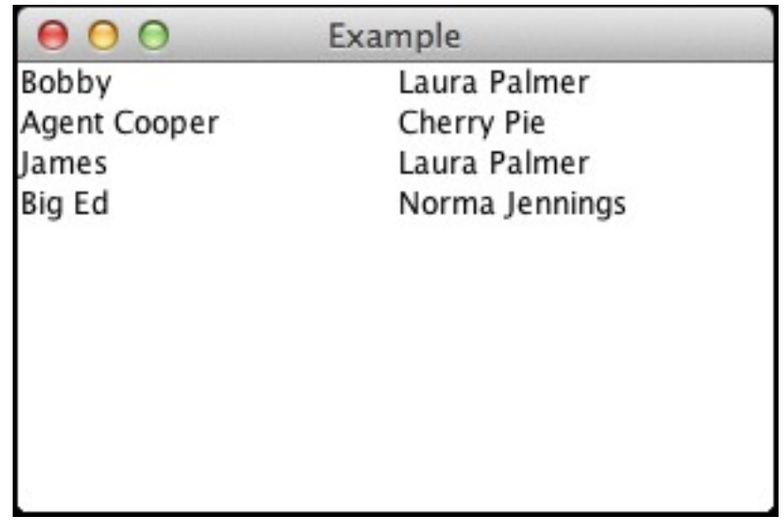
scrollable works for most widgets

Menus




```
(def sample-menu
  (seesaw/menubar
    :items [(seesaw/menu :text "File"
                         :items [(seesaw/action :name "Open..."
                                                :key "menu O"
                                                :handler (fn [e] (println "Open something")))])
            (seesaw/menu :text "Edit"
                         :items [(seesaw/action :name "Undo"
                                                :key "menu Z"
                                                :handler (fn [e] (println "Undo something")))])]))

(-> (seesaw/frame :title "Sample"
                  :menubar sample-menu
                  :content "Sampler"
                  :size [100 :by 50]
                   )
    seesaw/show!)
```

Tables




```
(def table (seesaw/table
  :model [
    :columns [{:key :name, :text "Name"} :likes]
    :rows [["Bobby" "Laura Palmer"]
           ["Agent Cooper" "Cherry Pie"]
           {:likes "Laura Palmer" :name "James"}
           {:name "Big Ed" :likes "Norma Jennings"}]]))
```

```
(display table 300 200)
```

Columns & Rows

```
:columns [{:key :name, :text "Name"} :likes]
:rows [["Bobby" "Laura Palmer"]
       ["Agent Cooper" "Cherry Pie"]
       {:likes "Laura Palmer" :name "James"}
       {:name "Big Ed" :likes "Norma Jennings"}]]))
```

Column :key

In example keys are :name & :likes

Used to get data from row maps

If row is vector then use position

Columns & Rows

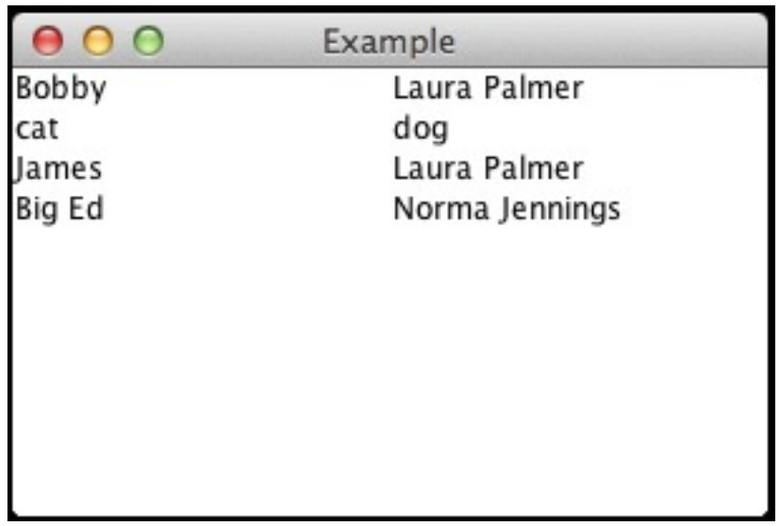
```
:columns [{:key :name, :text "Name"} :likes]
:rows [["Bobby" "Laura Palmer"]
       ["Agent Cooper" "Cherry Pie"]
       {:likes "Laura Palmer" :name "James"}
       {:name "Big Ed" :likes "Norma Jennings"}]]))
```

Column :text

In example text are “Names & likes

Used for column headers

Accessing & Changing Rows

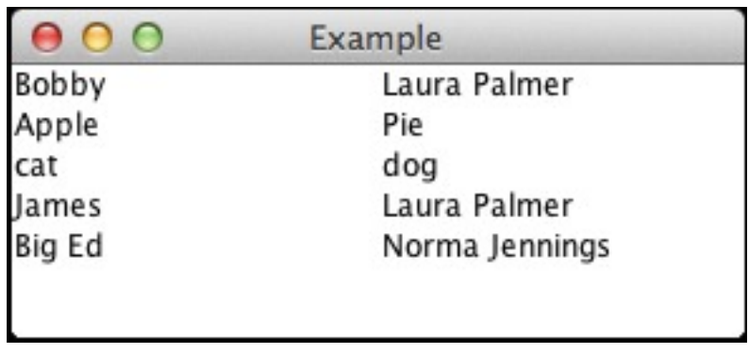



```
(def table (seesaw/table
  :model [
    :columns [{:key :name, :text "Name"} :likes]
    :rows [["Bobby" "Laura Palmer"]
           ["Agent Cooper" "Cherry Pie"]
           {:likes "Laura Palmer" :name "James"}
           {:name "Big Ed" :likes "Norma Jennings"}]]))
```

```
(seesaw.table/value-at table 1)          {:likes "Cherry Pie", :name "Agent Cooper"}

(seesaw.table/update-at! table 1 ["cat" "dog"])

(seesaw.table/value-at table 1)          {:likes "dog", :name "cat"}
```

Accessing & Changing Rows

```
(seesaw.table/insert-at! table 1 ["Apple" "Pie"])
```




```
(seesaw.table/remove-at! table 0)
```