FrTime: A Language for Reactive Programs

Version 9.4.0.4

Greg Cooper

November 3, 2026

`#lang frtime` package: `frtime`

The `frtime` language supports declarative construction of reactive systems in a syntax very similar to that of Racket. It extends `racket`.

Within DrRacket, as an alternative to using `#lang frtime`, you can choose FrTime from the Choose Language menu.

This reference document describes the functionality provided by the language; for details of the design and semantic model, consult the accompanying research papers:

- Brown 2008 - Cooper. Integrating dataflow evaluation into a practical higher-order call-by-value language
- ESOP 2006 — Cooper and Krishnamurthi. Embedding Dynamic Dataflow in a Call-by-Value Language
- Brown 2004 - Cooper and Krishnamurthi. FrTime : Functional Reactive Programming in PLT Scheme

1 Primitives

```
undefined : any/c
```

stands for an undefined value.

```
(undefined? val) → boolean?
  val : any/c
```

return #t iff *val* is undefined.

```
(behavior? val) → boolean?
  val : any/c
```

returns #t iff *val* is a behavior (a time-varying value whose current value can be projected at any time).

```
(event? val) → boolean?
  val : any/c
```

returns #t iff *val* is an event (a time-varying stream of values that can occur at arbitrary times).

```
(signal? val) → boolean?
  val : any/c
```

returns #t iff val is a signal. (signal? v) is equivalent to (or (behavior? v) (event? v)).

```
seconds : behavior?
```

updates approximately once per second with the value of (current-seconds).

```
milliseconds : behavior?
```

updates frequently with the value of (current-inexact-milliseconds).

```
never-e : event?
```

is an event that never occurs.

2 Defining Custom Input Signals

```
(new-cell [init-expr]) → signal?
  init-expr : signal? = undefined
```

returns a signal whose values initially track that of *init-expr*, but that may be rewired to a different signal by `set-cell!`.

```
(set-cell! cell val) → void?
  cell : signal?
  val : signal?
```

rewires *cell* (which must have been created by `new-cell`) to take on the value(s) of *val*.

```
(event-receiver) → event?
```

returns an event stream that can be triggered imperatively by `send-event`.

```
(send-event rcvr val) → void?
  rcvr : event?
  val : any/c
```

emits *val* on *rcvr* (which must have been created by `event-receiver`).

3 Signal-Processing Procedures

```
(value-now val) → any/c
  val : any/c
```

projects the current value of a behavior or constant.

```
(delay-by val duration) → behavior?
  val : behavior?
  duration : number?
```

delays *val* by *duration* milliseconds.

```
(integral val) → behavior?
  val : (or/c number? behavior?)
```

computes a numeric approximation of the integral of *val* with respect to time (measured in milliseconds).

```
(derivative val) → behavior?
  val : behavior?
```

computes a numeric approximation of the derivative of *val* with respect to time.

```
(map-e proc ev) → event?
  proc : (-> any/c any)
  ev : event?
(==> ev proc) → event?
  ev : event?
  proc : (-> any/c any)
```

returns an event stream that fires whenever *ev* fires, whose values are transformed by application of *proc*.

```
(filter-e pred ev) → event?
  pred : (-> any/c boolean?)
  ev : event?
(=#> ev pred) → event?
  ev : event?
  pred : (-> any/c boolean?)
```

returns an event stream that passes through only the values from *ev* for which *pred* returns #t.

```
(merge-e ev ...) → event?
  ev : event?
```

merges all of the input event sources into a single event source.

```
(once-e ev) → event?
  ev : event?
```

returns an event source that carries only the first occurrence of *ev*. (The rest are filtered out.)

```
(changes val) → event?
  val : behavior?
```

returns an event source that occurs each time the argument behavior changes. The value of the occurrence is the behavior's new value.

```
(hold ev [init]) → behavior?
  ev : event?
  init : any/c = undefined
```

constructs a behavior that starts out as *init* and then takes on the last value produced by *ev*

```
(switch ev [init]) → behavior?
  ev : event?
  init : behavior? = undefined
```

returns a behavior that starts as *init*. Each time *ev* yields a (potentially time-varying) value, the behavior switches to that value.

```
(accum-e ev init) → event?
  ev : event?
  init : any/c
```

constructs an event source by accumulating changes (carried by the given event source) over an initial value.

```
(accum-b ev init) → behavior?
  ev : event?
  init : any/c
```

combines functionality from `accum-e` and `hold` to construct a behavior. `(accum-b ev init)` is equivalent to `(hold init (accum-e ev init))`.

```
(collect-e ev init proc) → event?
  ev : event?
  init : any/c
  proc : (-> any/c any/c
             any)
```

is similar to `accum-e`, except the transformer function is fixed and is applied to the event occurrence and the current accumulator (in that order).

```
(collect-b ev init proc) → behavior?
  ev : event?
  init : any/c
  proc : (-> any/c any/c any)
```

is similar to `collect-e` in the same way as `accum-b` is similar to `accum-e`.

```
(when-e val) → event?
  val : behavior?
```

returns an event stream that carries an occurrence each time *val* changes from `#f` to anything else.

```
(lift-strict proc val ...) → any
  proc : (-> [arg any/c] ... any)
  val : any/c
```

provides a mechanism for applying ordinary Racket primitives to behaviors. If any of the *val*s are behaviors, returns a behavior whose current value is always equal to `(`*proc* `(value-now arg) ...)`. In FrTime, many Racket primitives are implicitly lifted.

The following forms allow importation of lifted procedures that aren't included in the basic FrTime language.

```
(require (lifted module-spec proc-name ...) ...)
(require (lifted:nonstrict module-spec proc-name ...) ...)
```

4 Fred: Functional Reactive Wrapper around GRacket

```
(require frtime/gui/fred)
```
package: frtime

```
ft-frame% : class?
  superclass: frame%
  extends: top-level-window<%>
```

```
(new ft-frame%
     [label label]
    [[parent parent]
     [width width]
     [height height]
     [x x]
     [y y]
     [style style]
     [enabled enabled]
     [border border]
     [spacing spacing]
     [alignment alignment]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]
     [shown shown]])
→ (is-a?/c ft-frame%)
 label : (or/c label-string? behavior?)
 parent : (or/c (is-a?/c frame%) false/c) = #f
 width : (or/c (integer-in 0 10000) false/c) = #f
 height : (or/c (integer-in 0 10000) false/c) = #f
 x : (or/c (integer-in -10000 10000) false/c) = #f
 y : (or/c (integer-in -10000 10000) false/c) = #f
 style : (listof (one-of/c 'no-resize-border 'no-caption
                           'no-system-menu 'hide-menu-bar
                           'mdi-parent 'mdi-child
                           'toolbar-button 'float 'metal))
       = null
 enabled : any/c = #t
 border : (integer-in 0 1000) = 0
 spacing : (integer-in 0 1000) = 0
 alignment : (list/c (one-of/c 'left 'center 'right)
                     (one-of/c 'top 'center 'bottom))
           = '(center top)
 min-width : (integer-in 0 10000) = graphical-minimum-width
```

```
  min-height : (integer-in 0 10000) = graphical-minimum-height
  stretchable-width : any/c = #t
  stretchable-height : any/c = #t
  shown : any/c = #f
```

The constructor arguments are as in frame%, except that *shown label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying.

```
ft-message% : class?
  superclass: message%
  extends: control<%>
```

```
 (new ft-message%
      [label label]
      [parent parent]
     [[style style]
      [font font]
      [enabled enabled]
      [vert-margin vert-margin]
      [horiz-margin horiz-margin]
      [min-width min-width]
      [min-height min-height]
      [stretchable-width stretchable-width]
      [stretchable-height stretchable-height]])
 → (is-a?/c ft-message%)
  label : (or/c label-string? behavior? (is-a?/c bitmap%)
                (or-of/c 'app 'caution 'stop))
  parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                 (is-a?/c panel%) (is-a?/c pane%))
  style : (listof (one-of/c 'deleted)) = null
  font : (is-a?/c font%) = (racket normal-control-font)
  enabled : (or/c any/c behavior?) = #t
  vert-margin : (integer-in 0 1000) = 2
  horiz-margin : (integer-in 0 1000) = 2
  min-width : (integer-in 0 10000) = graphical-minimum-width
  min-height : (integer-in 0 10000) = graphical-minimum-height
  stretchable-width : any/c = #f
  stretchable-height : any/c = #f
```

The constructor arguments are the same as in message%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying.

```
ft-button% : class?
  superclass: button%
  extends: control<%>
```

```
(new ft-button%
     [label label]
     [parent parent]
    [[style style]
     [font font]
     [enabled enabled]
     [vert-margin vert-margin]
     [horiz-margin horiz-margin]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]])
 → (is-a?/c ft-button%)
  label : (or/c label-string? behavior (is-a?/c bitmap%))
  parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                 (is-a?/c panel%) (is-a?/c pane%))
  style : (one-of/c 'border 'deleted) = null
  font : (is-a?/c font%) = (racket normal-control-font)
  enabled : any/c = #t
  vert-margin : (integer-in 0 1000) = 2
  horiz-margin : (integer-in 0 1000) = 2
  min-width : (integer-in 0 10000) = graphical-minimum-width
  min-height : (integer-in 0 10000) = graphical-minimum-height
  stretchable-width : any/c = #f
  stretchable-height : any/c = #f
```

The constructor arguments are the same as in `message%`, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying.

```
(send a-ft-button get-value-e) → event?
```

returns an event stream that yields a value whenever the user clicks the button.

```
ft-check-box% : class?
  superclass: check-box%
  extends: control<%>
```

```
(new ft-check-box%
     [label label]
     [parent parent]
    [[style style]
     [value value]
     [font font]
     [enabled enabled]
     [vert-margin vert-margin]
     [horiz-margin horiz-margin]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]
     [value-set value-set]])
→ (is-a?/c ft-check-box%)
 label : (or/c label-string? behavior? (is-a?/c bitmap%))
 parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                (is-a?/c panel%) (is-a?/c pane%))
 style : (listof (one-of/c 'deleted)) = null
 value : any/c = #f
 font : (is-a?/c font%) = (racket normal-control-font)
 enabled : any/c = #t
 vert-margin : (integer-in 0 1000) = 2
 horiz-margin : (integer-in 0 1000) = 2
 min-width : (integer-in 0 10000) = graphical-minimum-width
 min-height : (integer-in 0 10000) = graphical-minimum-height
 stretchable-width : any/c = #f
 stretchable-height : any/c = #f
 value-set : event? = never-e
```

The constructor arguments are the same as in check-box%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying. Also, any occurrence on *value-set* sets the check box's state to that of the event value.

```
(send a-ft-check-box get-value-b) → behavior?
```

returns a value that always reflects the current state of the check box.

```
ft-slider% : class?
  superclass: slider%
  extends: control<%>
```

```
(new ft-slider%
     [label label]
     [min-value min-value]
     [max-value max-value]
     [parent parent]
    [[init-value init-value]
     [style style]
     [font font]
     [enabled enabled]
     [vert-margin vert-margin]
     [horiz-margin horiz-margin]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]
     [value-set value-set]])
→ (is-a?/c ft-slider%)
 label : (or/c label-string? behavior? false/c)
 min-value : (integer-in -10000 10000)
 max-value : (integer-in -10000 10000)
 parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                (is-a?/c panel%) (is-a?/c pane%))
 init-value : (integer-in -10000 10000) = min-value
 style : (listof (one-of/c 'horizontal 'vertical 'plain
                           'vertical-label 'horizontal-label
                           'deleted))
       = '(horizontal)
 font : (is-a?/c font%) = normal-control-font
 enabled : any/c = #t
 vert-margin : (integer-in 0 1000) = 2
 horiz-margin : (integer-in 0 1000) = 2
 min-width : (integer-in 0 10000) = graphical-minimum-width
 min-height : (integer-in 0 10000) = graphical-minimum-height
 stretchable-width : any/c = (memq 'horizontal style)
 stretchable-height : any/c = (memq 'vertical style)
 value-set : event? = never-e
```

The constructor arguments are the same as in check-box%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying. Also, any occurrence on *value-set* sets the slider's state to that of the event value.

```
(send a-ft-slider get-value-b) → behavior?
```

returns a value that always reflects the current state of the slider.

```
ft-text-field% : class?
  superclass: text-field%
  extends: control<%>
```

```
(new ft-text-field%
     [label label]
     [parent parent]
    [[init-value init-value]
     [style style]
     [font font]
     [enabled enabled]
     [vert-margin vert-margin]
     [horiz-margin horiz-margin]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]
     [value-set value-set]])
 → (is-a?/c ft-text-field%)
  label : (or/c label-string? false/c)
  parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                 (is-a?/c panel%) (is-a?/c pane%))
  init-value : string? = ""
  style : (listof (one-of/c 'single 'multiple 'hscroll 'password
                            'vertical-label 'horizontal-label
                            'deleted))
        = '(single)
  font : (is-a?/c font%) = (racket normal-control-font)
  enabled : any/c = #t
  vert-margin : (integer-in 0 1000) = 2
  horiz-margin : (integer-in 0 1000) = 2
  min-width : (integer-in 0 10000) = graphical-minimum-width
  min-height : (integer-in 0 10000) = graphical-minimum-height
  stretchable-width : any/c = #t
  stretchable-height : any/c = (memq 'multiple style)
  value-set : event? = never-e
```

The constructor arguments are the same as in check-box%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying. Also, any occurrence on *value-set* sets the text field's state to that of the event value.

```
(send a-ft-text-field get-value-b) → behavior?
```

returns a value that always reflects the current state of the text field.

```
ft-radio-box% : class?
  superclass: radio-box%
  extends: control<%>
```

```
(new ft-radio-box%
     [label label]
     [choices choices]
     [parent parent]
    [[style style]
     [selection selection]
     [font font]
     [enabled enabled]
     [vert-margin vert-margin]
     [horiz-margin horiz-margin]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]
     [value-set value-set]])
→ (is-a?/c ft-radio-box%)
 label : (or/c label-string? behavior? false/c)
 choices : (or/c (listof label-string?) (listof (is-a?/c bitmap%)))
 parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                (is-a?/c panel%) (is-a?/c pane%))
 style : (listof (one-of/c 'horizontal 'vertical
                           'vertical-label 'horizontal-label
                           'deleted))
       = '(vertical)
 selection : exact-nonnegative-integer? = 0
 font : (is-a?/c font%) = normal-control-font
 enabled : any/c = #t
 vert-margin : (integer-in 0 1000) = 2
 horiz-margin : (integer-in 0 1000) = 2
 min-width : (integer-in 0 10000) = graphical-minimum-width
 min-height : (integer-in 0 10000) = graphical-minimum-height
 stretchable-width : any/c = #f
 stretchable-height : any/c = #f
 value-set : event? = never-e
```

The constructor arguments are the same as in check-box%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-

varying. Also, any occurrence on *value-set* sets the text field's state to that of the event value.

```
(send a-ft-radio-box get-selection-b) → behavior?
```

returns a value that always reflects the currently selected element in the radio box.

```
ft-choice% : class?
  superclass: choice%
  extends: control<%>
```

```
 (new ft-choice%
    [label label]
    [choices choices]
    [parent parent]
   [[style style]
    [selection selection]
    [font font]
    [enabled enabled]
    [vert-margin vert-margin]
    [horiz-margin horiz-margin]
    [min-width min-width]
    [min-height min-height]
    [stretchable-width stretchable-width]
    [stretchable-height stretchable-height]
    [value-set value-set]])
 → (is-a?/c ft-choice%)
  label : (or/c label-string? false/c)
  choices : (listof label-string?)
  parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                 (is-a?/c panel%) (is-a?/c pane%))
  style : (listof (one-of/c 'horizontal-label 'vertical-label
                            'deleted))
        = null
  selection : exact-nonnegative-integer? = 0
  font : (is-a?/c font%) = (racket normal-control-font)
  enabled : any/c = #t
  vert-margin : (integer-in 0 1000) = 2
  horiz-margin : (integer-in 0 1000) = 2
  min-width : (integer-in 0 10000) = graphical-minimum-width
  min-height : (integer-in 0 10000) = graphical-minimum-height
  stretchable-width : any/c = #f
```

```
  stretchable-height : any/c = #f
  value-set : event? = never-e
```

The constructor arguments are the same as in check-box%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying. Also, any occurrence on *value-set* sets the text field's state to that of the event value.

```
(send a-ft-choice get-selection-b) → behavior?
```

returns a value that always reflects the currently selected element in the choice control.

```
ft-list-box% : class?
  superclass: list-box%
  extends: control<%>
```

```
 (new ft-list-box%
     [label label]
     [choices choices]
     [parent parent]
    [[style style]
     [selection selection]
     [font font]
     [label-font label-font]
     [enabled enabled]
     [vert-margin vert-margin]
     [horiz-margin horiz-margin]
     [min-width min-width]
     [min-height min-height]
     [stretchable-width stretchable-width]
     [stretchable-height stretchable-height]
     [value-set value-set]])
 → (is-a?/c ft-list-box%)
  label : (or/c label-string? false/c)
  choices : (listof label-string?)
  parent : (or/c (is-a?/c frame%) (is-a?/c dialog%)
                 (is-a?/c panel%) (is-a?/c pane%))
  style : (listof (one-of/c 'single 'multiple 'extended
                            'vertical-label 'horizontal-label
                            'deleted))
        = '(single)
  selection : (or/c exact-nonnegative-integer? false/c) = #f
```

```
font : (is-a?/c font%) = (racket view-control-font)
label-font : (is-a?/c font%) = (racket normal-control-font)
enabled : any/c = #t
vert-margin : (integer-in 0 1000) = 2
horiz-margin : (integer-in 0 1000) = 2
min-width : (integer-in 0 10000) = graphical-minimum-width
min-height : (integer-in 0 10000) = graphical-minimum-height
stretchable-width : any/c = #t
stretchable-height : any/c = #t
value-set : event? = never-e
```

The constructor arguments are the same as in check-box%, except that *label*, *enabled*, *stretchable-width*, and *stretchable-height* may be time-varying. Also, any occurrence on *value-set* sets the text field's state to that of the event value.

```
(send a-ft-list-box get-selection-b) → behavior?
```

returns a value that always reflects the primary selection in the list box.

```
(send a-ft-list-box get-selections-b) → behavior?
```

returns a value that always reflects the current set of selected elements in the list box.

5 Graphical Demo Programs

TODO: document the animation library itself!

To run the following animation/GUI demos, simply set the language level to FrTime, open the corresponding file, and Execute. See the demo source code for more information.

`"orbit-mouse.rkt"` : A collection of balls that move in circles around the mouse pointer.

`"piston.rkt"` : Simulation of a piston/cylinder.

`"rotation.rkt"` : Balls moving in circles.

`"delay-mouse.rkt"` : A trail of balls following the mouse.

`"ball-on-string.rkt"` : A ball chasing the mouse.

`"pong.rkt"` : A simple pong/air-hockey game. The left paddle moves with numeric keypad; the right paddle moves with the mouse. The 'r' key resets the score.

`"pizza.rkt"` : A simple "pizza ordering" user interface based on an HtDP exercise.

`"calculator.rkt"` : A simple calculator interface, also based on an HtDP exercise except that the result updates continuously as the arguments and operator change.

The next three animation examples are courtesy of Robb Cutler:

`"analog-clock.rkt"` : An animated real-time clock. A slider adjusts the radius of the face. Click and drag to move the face around.

`"growing-points.rkt"` : A field of points that grow as the mouse approaches.

`"needles.rkt"` : A field of needles that point at the mouse.