Images

Version 5.2.1

Neil Toronto <neil.toronto@gmail.com>

February 2, 2012

This library contains convenient functions for constructing icons and logos, and will eventually offer the same for other `bitmap%`s. The idea is to make it easy to include such things in your own programs.

Generally, the images in this library are computed when requested, not loaded from disk. Most of them are drawn on a `dc<%>` and then ray traced. This can become computationally expensive, so this library also includes `images/compile-time`, which makes it easy to compute images at compile time and access them at run time.

Contents

1 Icons

1.1 What is an icon?

This section describes an ideal that DrRacket and its tools are steadily approaching.

As a first approximation, an icon is just a small `bitmap%`, usually with an alpha channel.

But an icon also communicates. Its shape and color are a visual metaphor for an action or a message. Icons should be **easily recognizable**, **distinguishable**, **visually consistent**, and **metaphorically appropriate** for the actions and messages they are used with. It can be difficult to meet all four requirements at once ("distinguishable" and "visually consistent' are often at odds), but good examples, good abstractions, and an existing icon library help considerably.

Example: The Macro Stepper icon is composed by appending a text icon and a step icon to get . The syntax quote icon is the color that DrRacket colors syntax quotes by default. The step icon is colored like DrRacket colors identifier syntax by default, and is shaped using metaphors used in debugger toolbars, TV remotes, and music players around the world. It is composed of to connote starting and to connote immediately stopping.

It would not do to have just as the Macro Stepper icon: it would be too easily confused with the Debugger icon , especially for new users and people with certain forms of color-blindness, and thus fail to be distinguishable enough.

As another example, the Check Syntax icon connotes inspecting and passing. Note that the check mark is also the color of syntax.

1.2 About These Icons

The icons in this collection are designed to be composed to create new ones: they are simple, thematically consistent, and can be constructed in any size and color. Further, slideshow's `pict` combiners offer a way to compose them almost arbitrarily. For example, a media player application might create a large "step" button by superimposing a `record-icon` and a `step-icon`:

```
> (require slideshow/pict images/icons/control images/icons/style)

> (pict->bitmap
   (cc-superimpose
    (bitmap (record-icon "forestgreen" 96 glass-icon-material))
    (bitmap (step-icon light-metal-icon-color 48 metal-icon-
material))))
```

All the icons in this collection are first drawn using standard `dc<%>` drawing commands. Then, to get lighting effects, they are turned into 3D objects and ray traced. Many are afterward composed to create new icons; for example, the `stop-signs-icon` superimposes three `stop-sign-icon`s, and the `magnifying-glass-icon` is composed of three others (frame, glass and handle).

The ray tracer helps keep icons visually consistent with each other and with physical objects in day-to-day life. As an example of the latter, the `record-icon`, when rendered in clear glass, looks like the clear, round button on a Wii Remote. See the `plt-logo` and `planet-logo` functions for more striking examples.

When the rendering API is stable enough to publish, it will allow anyone who can draw a shape to turn that shape into a visually consistent icon.

As with any sort of rendering (such as SVG rendering), ray tracing takes time. For icons, this usually happens during tool or application start up. You can reduce the portion of start-up time taken by rendering to almost nothing by using the `images/compile-time` library to embed bitmaps directly into compiled modules.

1.3 Icon Style

```
(require images/icons/style)
```

Use these constants and parameters to help keep icon sets visually consistent.

```
light-metal-icon-color : (or/c string? (is-a?/c color%))
= "azure"
```

```
metal-icon-color : (or/c string? (is-a?/c color%))
```

```
= "lightsteelblue"
```

```
dark-metal-icon-color : (or/c string? (is-a?/c color%))
= "steelblue"
```

Good colors to use with `metal-icon-material`. See `bomb-icon` and `magnifying-glass-icon` for examples.

```
syntax-icon-color : (or/c string? (is-a?/c color%))
= (make-object color% 76 76 255)
```

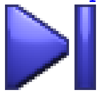
```
halt-icon-color : (or/c string? (is-a?/c color%))
= (make-object color% 255 32 24)
```

```
run-icon-color : (or/c string? (is-a?/c color%))
= "lawngreen"
```

Standard toolbar icon colors.

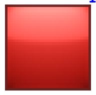
Use `syntax-icon-color` in icons that connote macro expansion or syntax. Example:

```
> (step-icon syntax-icon-color 32)
```



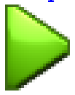
Use `halt-icon-color` in icons that connote stopping or errors. Example:

```
> (stop-icon halt-icon-color 32)
```



Use `run-icon-color` in icons that connote executing programs or evaluation. Examples:

```
> (play-icon run-icon-color 32)
```



```
> (require images/icons/stickman)
> (running-stickman-icon 0.9 run-icon-color "white" run-icon-
color 32)
```

For new users and for accessibility reasons, do not try to differentiate icons for similar functions only by color.

```
(default-icon-height) → (and/c rational? (>=/c 0))
(default-icon-height height) → void?
  height : (and/c rational? (>=/c 0))

= 24
```

The height of DrRacket's standard icons.

```
(toolbar-icon-height) → (and/c rational? (>=/c 0))
(toolbar-icon-height height) → void?
  height : (and/c rational? (>=/c 0))

= 16
```

The height of DrRacket toolbar icons.

Use `(toolbar-icon-height)` as the `height` argument for common icons that will be used in toolbars, status bars, and buttons.

(When making an icon for DrRacket's main toolbar, try to keep it nearly square so that it will not take up too much horizontal space when the toolbar is docked vertically. If you cannot, as with the Macro Stepper, send a thinner icon as the `alternate-bitmap` argument to a `switchable-button%`.)

```
plastic-icon-material : deep-flomap-material-value?
```

```
glass-icon-material : deep-flomap-material-value?
```

```
metal-icon-material : deep-flomap-material-value?
```

Materials for icons.

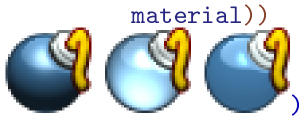
Plastic is opaque and reflects a little more than glass.

Glass is transparent but frosted, so it scatters refracted light. It has the high refractive index of cubic zirconia, or fake diamond. The "glassy look" cannot actually be achieved using glass.

Metal reflects the most, its specular highlight is nearly the same color as the material (in the others, the highlight is white), and it diffuses much more ambient light than directional. This is because, while plastic and glass mostly reflect light directly, metal mostly absorbs light and re-emits it.

Examples:

```
> (require images/icons/misc)

> (for/list ([material  (list plastic-icon-material
                              glass-icon-material
                              metal-icon-material)])
    (bomb-icon light-metal-icon-color dark-metal-icon-color 32
               material))
```

(list )

```
(default-icon-material) → deep-flomap-material-value?
(default-icon-material material) → void?
  material : deep-flomap-material-value?

= plastic-icon-material
```

The material used for rendering most icons and icon parts. There are exceptions; for example, the `floppy-disk-icon` always renders the sliding cover in metal.

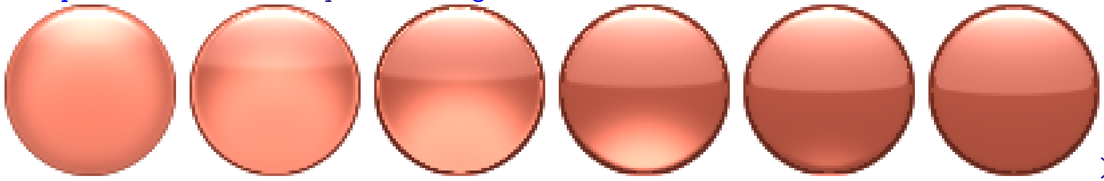
```
(bitmap-render-icon bitmap [z-ratio material]) → (is-a?/c bitmap%)
  bitmap : (is-a?/c bitmap%)
  z-ratio : (and rational? (>=/c 0)) = 5/8
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Makes a 3D object out of *bitmap* and renders it as an icon.

The *z-ratio* argument only makes a difference when *material* is transparent, such as `glass-icon-material`. It controls what fraction of *bitmap*'s height the icon is raised, which in turn affects the refracted shadow under the icon: the higher the *z-ratio*, the lower the shadow.

Examples:

```
> (define bitmap
    (pict->bitmap (colorize (filled-ellipse 64 64) "tomato")))
```

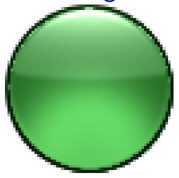
```
> (for/list ([z-ratio  (in-range 0 2 1/3)])
    (bitmap-render-icon bitmap z-ratio glass-icon-material))
```
(list )

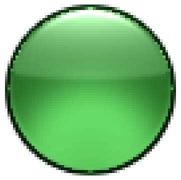
More complex shapes than “embossed and rounded” are possible with the full rendering API, which will be made public in a later release. Still, most of the simple icons (such as those in `images/icons/arrow` and `images/icons/control`) can be rendered using only `bitmap-render-icon`.

```
(icon-color->outline-color color) → (is-a?/c color%)
  color : (or/c string? (is-a?/c color%))
```

For a given icon color, returns the proper outline `color%`.

As an example, here is how to duplicate the `record-icon` using `slideshow/pict`:

```
> (define outline-color (icon-color->outline-color "forestgreen"))

> (define brush-pict (colorize (filled-ellipse 62 62) "forestgreen"))

> (define pen-pict (linewidth 2 (colorize (ellipse 62 62) outline-
color)))

> (bitmap-render-icon
   (pict->bitmap (inset (cc-superimpose brush-pict pen-pict) 1))
   5/8 glass-icon-material)
```


```
> (record-icon "forestgreen" 64 glass-icon-material)
```


The outline width is usually `(/ height 32)` (in this case, `2`), but not always. (For example, recycle-icon is an exception, as are parts of floppy-disk-icon.)

1.4 Arrow Icons

```
(require images/icons/arrow)
```

```
(right-arrow-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(left-arrow-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

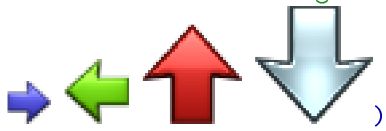
```
(up-arrow-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(down-arrow-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Standard directional arrows.

Example:

```
> (list (right-arrow-icon syntax-icon-color (toolbar-icon-height))
        (left-arrow-icon run-icon-color)
        (up-arrow-icon halt-icon-color 37)
        (down-arrow-icon "lightblue" 44 glass-icon-material))
```

(list )

```
(right-over-arrow-icon  color
                       [height
                        material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(left-over-arrow-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

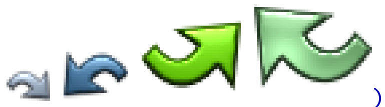
```
(right-under-arrow-icon  color
                        [height
                         material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(left-under-arrow-icon  color
                       [height
                        material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Standard bent arrows.

Example:

```
> (list (right-over-arrow-icon metal-icon-color (toolbar-icon-
height))
        (left-over-arrow-icon dark-metal-icon-color)
        (right-under-arrow-icon run-icon-color 37)
        (left-under-arrow-icon "lightgreen" 44 glass-icon-
material))
```

(list )

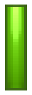
1.5 Control Icons

```
(require images/icons/control)
```

```
(bar-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (bar-icon run-icon-color 32)
```

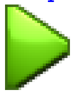


This is not a “control” icon *per se*, but is used to make many others.

```
(play-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

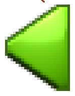
```
> (play-icon run-icon-color 32)
```



```
(back-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

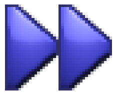
```
> (back-icon run-icon-color 32)
```



```
(fast-forward-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

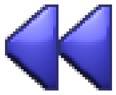
```
> (fast-forward-icon syntax-icon-color 32)
```



```
(rewind-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

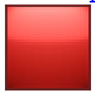
```
> (rewind-icon syntax-icon-color 32)
```



```
(stop-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

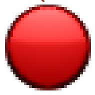
```
> (stop-icon halt-icon-color 32)
```



```
(record-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

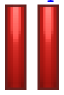
```
> (record-icon "red" 32)
```



```
(pause-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

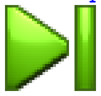
```
> (pause-icon halt-icon-color 32)
```



```
(step-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

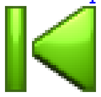
```
> (step-icon run-icon-color 32)
```



```
(step-back-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

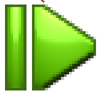
```
> (step-back-icon run-icon-color 32)
```



```
(continue-forward-icon  color
                       [height
                        material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

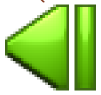
```
> (continue-forward-icon run-icon-color 32)
```



```
(continue-backward-icon  color
                        [height
                         material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

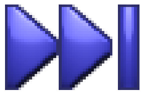
```
> (continue-backward-icon run-icon-color 32)
```



```
(search-forward-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

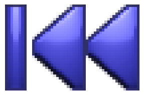
```
> (search-forward-icon syntax-icon-color 32)
```



```
(search-backward-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (search-backward-icon syntax-icon-color 32)
```



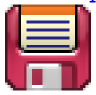
1.6 File Icons

```
(require images/icons/file)
```

```
(floppy-disk-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

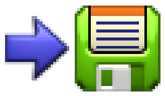
```
> (floppy-disk-icon "crimson" 32 glass-icon-material)
```



```
(save-icon arrow-color color [height material]) → (is-a?/cbitmap%)
  arrow-color : (or/c string? (is-a?/c color%))
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

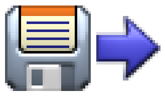
```
> (save-icon syntax-icon-color run-icon-color 32)
```



```
(load-icon arrow-color color [height material]) → (is-a?/c bitmap%)
  arrow-color : (or/c string? (is-a?/c color%))
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

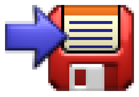
```
> (load-icon syntax-icon-color metal-icon-color 32)
```



```
(small-save-icon  arrow-color
                  color
                 [height
                  material])  → (is-a?/c bitmap%)
  arrow-color : (or/c string? (is-a?/c color%))
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

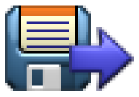
```
> (small-save-icon syntax-icon-color halt-icon-color 32)
```



```
(small-load-icon  arrow-color
                  color
                 [height
                  material])  → (is-a?/c bitmap%)
  arrow-color : (or/c string? (is-a?/c color%))
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (small-load-icon syntax-icon-color dark-metal-icon-color 32)
```



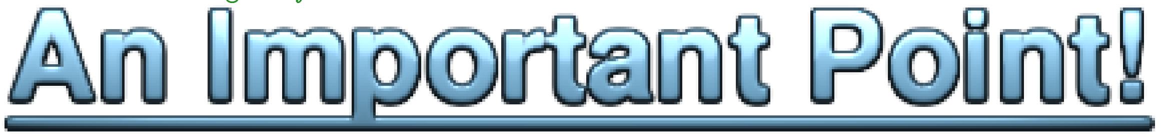
1.7 Symbol and Text Icons

```
(require images/icons/symbol)
```

```
(text-icon  str
            font
            color
           [trim?
            outline
            height
            material]) → (is-a?/c bitmap%)
  str : string?
  font : (is-a?/c font%)
  color : (or/c string? (is-a?/c color%))
  trim? : boolean? = #t
  outline : (or/c 'auto (and/c rational? (>=/c 0))) = 'auto
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

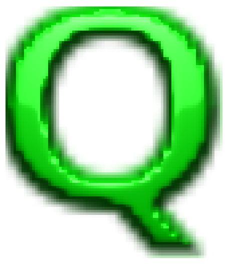
Renders a text string as an icon. For example,

```
> (text-icon "An Important Point!"
             (make-object font% 48 'decorative 'normal 'bold #t)
             "lightskyblue" #t 'auto 48)
```



Before rendering, the drawn text is scaled so that it is exactly *height* pixels tall. Make sure the font is large enough that scaling does not create blurry and jagged edge artifacts, as in the following example:

```
> (text-icon "Q" (make-object font% 32 'default 'normal 'bold)
             "green" #t 0 96)
```

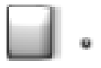


When *str* contains tall letters or *trim?* is `#f`, using *height* as the font size should be sufficient.

To make it easy to create a large enough font, `text-icon` always interpets font sizes as being in pixels, never points. See `font%` for details on font sizes.

If *trim?* is `#f`, the drawn text is not cropped before rendering. Otherwise, it is cropped to the smallest rectangle containing all the non-zero-alpha pixels. Rendering very small glyphs shows the difference dramatically:

```
> (define font (make-object font% 32 'default))

> (list (text-icon "." font "white")
        (text-icon "." font "white" #f))
```

```
(list
```

```
)
```

Note that both icons are `(default-icon-height)` pixels tall.

When *outline* is `'auto`, the outline drawn around the text is `(/ height 32)` pixels wide.

Because different platforms have different fonts, `text-icon` cannot guarantee the icons it returns have a consistent look or width across all platforms, or that the unicode characters will exist.

```
(recycle-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Returns the universal recycling symbol, rendered as an icon.

Example:

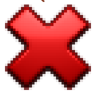
```
> (recycle-icon (make-object color% 0 153 0) 48)
```

```
(x-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Returns an "x" icon that is guaranteed to look the same on all platforms. (Anything similar that would be constructed by `text-icon` would differ at least slightly across platforms.)

Example:

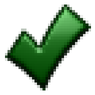
```
> (x-icon "red" 32)
```



```
(check-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

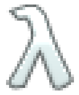
```
> (check-icon "darkgreen" 32)
```



```
(lambda-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

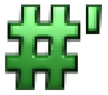
```
> (lambda-icon light-metal-icon-color 32 metal-icon-material)
```



```
(hash-quote-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Examples:

```
> (require (only-in images/icons/tool macro-stepper-hash-color))

> (hash-quote-icon macro-stepper-hash-color 32)
```



1.8 Miscellaneous Icons

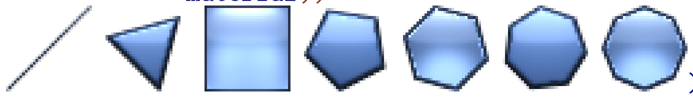
```
(require images/icons/misc)
```

```
(regular-polygon-icon  sides
                       start
                       color
                      [height
                       material]) → (is-a?/c bitmap%)
  sides : exact-positive-integer?
  start : real?
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Renders the largest regular polygon with *sides* sides, with the first vertex at angle *start*, that can be centered in a *height* × *height* box.

Example:

```
 > (for/list ([sides  (in-range 1 9)]
              [material  (in-cycle (list plastic-icon-material
                                          glass-icon-material))])
     (regular-polygon-icon sides (* 1/4 pi) "cornflowerblue" 32
                           material))
```

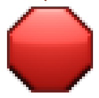
(list )

```
(octagon-icon color [height material]) → (is-a?/cbitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Equivalent to `(regular-polygon-icon 8 (/ (* 2 pi) 16) color height material)`.

Example:

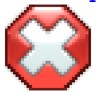
```
 > (octagon-icon halt-icon-color 32)
```



```
(stop-sign-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (stop-sign-icon halt-icon-color 32 glass-icon-material)
```



```
(stop-signs-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (stop-signs-icon halt-icon-color 32 plastic-icon-material)
```

```
(foot-icon color [height material]) → (is-a?/c bitmap%)
  color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (foot-icon "chocolate" 32 glass-icon-material)
```

```
(magnifying-glass-icon  frame-color
                        handle-color
                       [height
                        material])    → (is-a?/c bitmap%)
  frame-color : (or/c string? (is-a?/c color%))
  handle-color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (magnifying-glass-icon light-metal-icon-color "lightblue" 32
                         glass-icon-material)
```

```
(left-magnifying-glass-icon  frame-color
                             handle-color
                            [height
                             material])    → (is-a?/cbitmap%)
  frame-color : (or/c string? (is-a?/c color%))
  handle-color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (left-magnifying-glass-icon metal-icon-color "red" 32)
```

```
(bomb-icon  cap-color
            bomb-color
           [height
            material])  → (is-a?/cbitmap%)
  cap-color : (or/c string? (is-a?/c color%))
  bomb-color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (bomb-icon light-metal-icon-color "black" 32 glass-icon-
material)
```

```
(left-bomb-icon  cap-color
                 bomb-color
                [height
                 material])  → (is-a?/cbitmap%)
```

```
  cap-color : (or/c string? (is-a?/c color%))
  bomb-color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Example:

```
> (left-bomb-icon metal-icon-color dark-metal-icon-color 32)
```

```
(clock-icon [height
             face-color
             hand-color
             hours
             minutes])   → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  face-color :  (or/c string? (is-a?/c color%))
             = light-metal-icon-color
  hand-color : (or/c string? (is-a?/c color%)) = "firebrick"
  hours : (integer-in 0 11) = 1
  minutes : (real-in 0 60) = 33
```

Examples:

```
> (clock-icon 48)
```

```
> (clock-icon 48 "lightblue" "darkblue" 3 21)
```

1.9 Stickman Icons

```
(require images/icons/stickman)
```

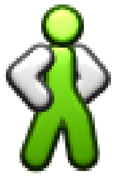
```
(standing-stickman-icon  color
                         arm-color
                         head-color
                        [height
                         material])  → (is-a?/c bitmap%)
```

```
  color : (or/c string? (is-a?/c color%))
  arm-color : (or/c string? (is-a?/c color%))
  head-color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Returns the icon displayed in DrRacket's lower-right corner when no program is running.

Example:

```
> (standing-stickman-icon run-icon-color "white" run-icon-
color 64)
```

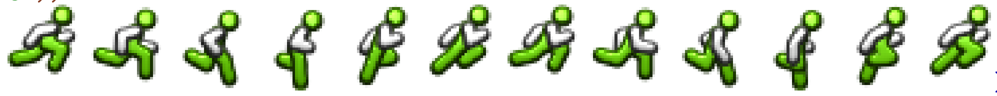


```
(running-stickman-icon  t
                        color
                        arm-color
                        head-color
                       [height
                        material])  → (is-a?/c bitmap%)
  t : rational?
  color : (or/c string? (is-a?/c color%))
  arm-color : (or/c string? (is-a?/c color%))
  head-color : (or/c string? (is-a?/c color%))
  height : (and/c rational? (>=/c 0)) = (default-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Returns a frame of the icon animated in DrRacket's lower-right corner when a program is running. The frame returned is for time *t* of a run cycle with a one-second period.

The following example samples the run cycle at 12 Hz, or every 1/12 second:

```
> (for/list ([t  (in-range 0 1 1/12)])
    (running-stickman-icon t run-icon-color "white" run-icon-
color 32))
```

(list )

The stickman's joint angles are defined by continuous periodic functions, so the run cycle can be sampled at any resolution, or at any real-valued time `t`. The cycle is modeled after the run cycle of the player's avatar in the Commodore 64 game Impossible Mission.

1.10 Tool Icons

```
(require images/icons/tool)
```

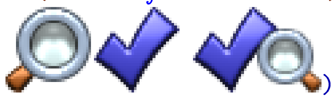
```
(check-syntax-icon [height material]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (toolbar-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(small-check-syntax-icon [height material]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (toolbar-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Icons for Check Syntax. The `small-check-syntax-icon` is used when the toolbar is on the side.

Example:

```
> (list (check-syntax-icon 32) (small-check-syntax-icon 32))
(list
```



```
)
```

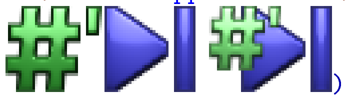
```
(macro-stepper-icon [height material]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (toolbar-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(small-macro-stepper-icon [height material]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (toolbar-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Icons for the Macro Stepper. The `small-macro-stepper-icon` is used when the toolbar is on the side.

Example:

```
> (list (macro-stepper-icon 32) (small-macro-stepper-icon 32))
```

(list )

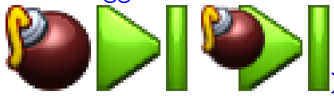
```
(debugger-icon [height material]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (toolbar-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

```
(small-debugger-icon [height material]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = (toolbar-icon-height)
  material : deep-flomap-material-value?
           = (default-icon-material)
```

Icons for the Debugger. The `small-debugger-icon` is used when the toolbar is on the side.

Example:

```
> (list (debugger-icon 32) (small-debugger-icon 32))
```

(list )

```
debugger-bomb-color : (or/c string? (is-a?/c color%))
= (make-object color% 128 32 32)
```

```
macro-stepper-hash-color : (or/c string? (is-a?/c color%))
= (make-object color% 60 192 60)
```

```
small-macro-stepper-hash-color : (or/c string? (is-a?/c color%))
= (make-object color% 128 255 128)
```

Constants used within `images/icons/tool`.

2 Logos

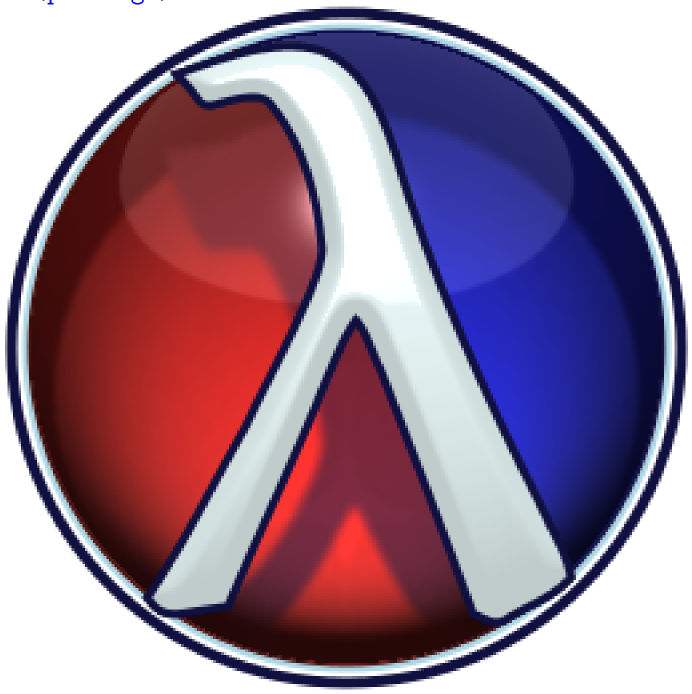
```
(require images/logos)
```

```
(plt-logo [height]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = 256
```

Returns the PLT logo, rendered in tinted glass and azure metal by the ray tracer that renders icons.

Example:

```
> (plt-logo)
```



The default height is the size used for DrRacket splash screen.

```
(planet-logo [height]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = 96
```

Returns an unofficial PLaneT logo. This is used as the PLaneT icon when DrRacket downloads PLaneT packages.

Examples:

```
> (planet-logo)
```

```
> (planet-logo (default-icon-height))
```

```
(stepper-logo [height]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = 96
```

Returns the algebraic stepper logo.

Example:

```
> (stepper-logo)
```

```
(macro-stepper-logo [height]) → (is-a?/c bitmap%)
  height : (and/c rational? (>=/c 0)) = 96
```

Returns the macro stepper logo.

Example:

```
> (macro-stepper-logo)
```

3 Embedding Bitmaps in Compiled Files

```
(require images/compile-time)
```

Producing computed bitmaps can take time. To reduce the startup time of programs that use computed bitmaps, use the macros exported by `images/compile-time` to *compile* them: to embed the computed bitmaps in fully expanded, compiled modules.

This is a form of constant folding, or equivalently a form of *safe* “3D” values.

The macros defined here compute bitmaps at expansion time, and expand to the bitmap’s `bytes` and a simple wrapper that converts `bytes` to a `bitmap%`. Thus, fully expanded, compiled modules contain (more or less) literal bitmap values, which do not need to be computed again when the module is `require`d by another.

The literal bitmap values are encoded in PNG format, so they are compressed in the compiled module.

To get the most from compiled bitmaps during development, it is best to put them in files that are changed infrequently. For example, for games, we suggest having a separate module called something like `images.rkt` or `resources.rkt` that `provide`s all the game’s images.

```
(compiled-bitmap expr)
```

Evaluates *`expr`* at expansion time, which must return a `bitmap%`, and returns to the bitmap at run time. Keep in mind that *`expr`* has access only to expansion-time values, not run-time values.

Generally, to use this macro, wrap a `bitmap%`-producing expression with it and move any identifiers it depends on into the expansion phase. For example, suppose we are computing a large PLT logo at run time:

```
#lang racket


(require images/logos)

(define the-logo (plt-logo 384))
```

Running this takes several seconds. It produces

```
> the-logo
```

To move the cost to expansion time, we change the program to

```
#lang racket

(require images/compile-time
         (for-syntax images/logos))

(define the-logo (compiled-bitmap (plt-logo 384)))
```

The logo is unchanged, but now *expanding* (and thus compiling) the program takes several seconds, and running it takes a few milliseconds. Note that `images/logos` is now required for-syntax, so that the expansion-phase expression (plt-logo 384) has access to the identifier plt-logo.

```
(compiled-bitmap-list expr)
```

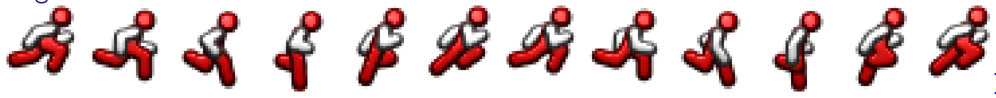
Like compiled-bitmap, but it expects *expr* to return a list of bitmap%s, and it returns the list at run time.

Use this for animations. For example,

```
#lang racket

(require images/compile-time
         (for-syntax images/icons/stickman))

(begin-for-syntax
  (define num-stickman-frames 12))

(define running-stickman-frames
  (compiled-bitmap-list
   (for/list ([t  (in-range 0 1 (/ 1 num-stickman-frames))])
     (running-stickman-icon t "red" "white" "red" 32))))
```

This computes

```
> running-stickman-frames
(list
```



```
)
```

at expansion time.