RebGUI Widget Designer's Guide

Author : Ashley G Truter
Updated: 20-Apr-2007
Purpose: Describes how to create RebGUI widgets.

---

1. Introduction

This document describes how to create RebGUI widgets. Go http://www.ross-gill.com/r/guides.htmlhere for a good visual summary of some of the more important UI design terminology and principles.

The simplest possible widget you can create is by editing the `widgets` context in `%rebgui-widgets.r` and adding a line like the following:

```
my-widget: make rebface []
```

But we'll cover a few more basics before returning to that.

2. The RebGUI face object

The standard RebGUI face definition is detailed below.

```
REBFACE is an object of value: [
    type            word!     face
    offset          pair!     0x0
    size            pair!     100x100
    span            none!     none
    pane            none!     none
    text            none!     none
    color           none!     none
    image           none!     none
    effect          none!     none
    data            none!     none
    edge            none!     none
    font            none!     none
    para            none!     none
    feel            object!   [
        redraw          none!     none
        detect          none!     none
        over            none!     none
        engage          none!     none
    ]
    saved-area      none!     none
    rate            none!     none
    show?           logic!    true
    options         block!    length: 0
    parent-face     none!     none
    old-offset      none!     none
    old-size        none!     none
    line-list       none!     none
    changes         none!     none
    face-flags      integer!  0
    action          object!   [
        on-alt-click    none!     none
        on-away         none!     none
        on-click        none!     none
        on-dbl-click    none!     none
        on-focus        none!     none
        on-key          none!     none
        on-over         none!     none
        on-scroll       none!     none
        on-unfocus      none!     none
    ]
    rebind          none!     none
    init            none!     none
    tip             none!     none
]
```

3. Available Facets

Of the 25 standard REBOL/View">http://www.rebol.com/docs/view-system.html#section-2.2">REBOL/View Facets only five are able to be used by RebGUI. These are detailed below.

3.1. Type

The `type` facet is set by the `display` function to the name of the widget. Do not redefine this facet.

3.2. Data

This facet may be freely used by the widget. If the widget makes use of a **single** non-standard attribute to control its behavior (e.g. the `progress` widget requires a decimal value to control its bar length) then use this facet for that purpose and create custom facets (see later) as needed.

3.3. Options

This facet is used to store specification flags and / or meta-data. It is always assumed to be a block.

3.4. Action

This facet is used to store the function associated with the `engage` feel. Do not redefine this facet.

3.5. Rebind

This facet contains a `function!` that is called when the `widgets/rebind` function is called, typically after UI settings have been changed via `request-ui`.

> Rebind is most often used to update attributes such as color that are bound when the `widgets` object is first created. For example:

```
color:    colors/widget
...
rebind: make function! [] [
    color: colors/widget
]
```

3.6. Init

This facet contains an initialization `function!` that is called after the widget spec has been processed but prior to the next widget. It is then set to `none`.

> If creating a widget that inserts another widget directly into a `pane` you may have to manually call its `init` function so it renders directly. For example:

```
insert tail pane make slider [
    tip: none
    ...
]
pane/1/init
```

3.7. Tip

This facet contains the Widget's `USAGE:`, `DESCRIPTION:` and `OPTIONS:` text. At runtime it is set to none or replaced with a display spec tip.

> If creating a widget that inserts another widget directly into a `pane` remember to set the widget's tip to `none!` otherwise the default help tip will be displayed at runtime. For example:

```
insert tail pane make arrow [
    tip: none
    ...
]
```

3.8. Custom Facets

Complex widgets may require more facets than those available; in this case you are encouraged to extend the face definition with your own custom facets.

4. Feel Function Templates

This section provides simple templates for the three main feel">http://www.rebol.com/docs/view-system.html#section-5.3">feel functions used in RebGUI widgets. The `detect` function is not covered as its use is discouraged.

4.1. Redraw

```
feel: make default-feel [
    ; pos is position of iterated face - ignore
    redraw: func [face act pos] [
        do select [
            draw []
            show []
            hide []
        ] act
    ]
]
```

4.2. Over

```
feel: make default-feel [
    ; pos is mouse position
    over: func [face into pos] [
        either into [] []
    ]
]
```

4.3. Engage

```
feel: make default-feel [
    engage: func [face act event] [
```

```
        do select [
            time     []
            move     []
            down     [either event/double-click [][]]
            up       []
            alt-down []
            alt-up   []
            over     []
            away     []
        ] act
    ]
]
```

5. The Edit object

This section describes the operation of the RebGUI edit feel used by widgets that accept keyboard input. It is based on the `%view-edit.r` SDK source and the subsequent work of Romano Paolo Tenca located http://www.rebol.it/~romano/edit-text-undo.txt">here.

The `behaviors` object has a number of properties that can be set to control the edit-related behavior of all widgets.

```
CTX-REBGUI/BEHAVIORS is an object of value:
    action-on-enter  block!    length: 5
    action-on-tab    block!    length: 1
    caret-on-focus   block!    length: 7
    cyclic           block!    length: 1
    hilight-on-focus block!    length: 4
    tabbed           block!    length: 9
```

In addition to this, a number of words in the `edit object!` itself are used within widgets.

```
make object! [
    insert?          logic!    true
    keymap           block!    length: 24
    feel             object!   [redraw detect over engage]
]
```

5.1. action-on-enter Block

Indicates whether the widget's `action` should be called when the **Enter** key is pressed.

5.2. action-on-tab Block

Indicates whether the widget's `action` should be called when the **Tab** key is pressed.

5.3. caret-on-focus Block

Indicates whether the caret should be set when the widget receives focus (either by mouse click or keyboard tab).

5.4. cyclic Block

Indicates whether the widgets within a grouping widget (e.g. tab-panel) are in a closed tabbing environment.

5.5. hilight-on-focus Block

When focus shifts to a widget in the `tabbed` block, either because it was clicked on or tabbed to, one of two things will happen: either the entire contents will be highlighted if the widget appears in the `hilight-on-focus` block, or the cursor will be placed at the end of its contents.

5.6. tabbed Block

The tabbed block contains a list of widgets which RebGUI can tab to and from.

> A tabbed widget will only receive focus if its `show?` attribute is set to `true`.

5.7. Insert? Flag

This is simply a `true` / `false` flag indicating whether `Insert` mode is on or off. This mode defaults to **on** and is toggled by pressing the `Ins` key (at which time the mode will be changed to "overwrite").

5.8. Keymap Block

This table contains default key bindings which you can modify as required. The default bindings are:

```
keymap: [
    #"^H" back-char
    #"^-" tab-char
    #"^~" del-char
    #"^M" enter
```

```
        #"^A" all-text
        #"^C" copy-text
        #"^X" cut-text
        #"^V" paste-text
        #"^T" clear-tail
        #"^Z" undo
        #"^Y" redo
        #"^[" undo-all
        #"^S" spellcheck
        #"^/" ctrl-enter
    ]
```

5.9. feel Object

The `feel` object is used in conjunction with the `font` and `para` objects to enable text editing. A basic implementation would be:

```
append-widget [
    my-widget: make ctx-rebgui/rebface [
        font: default-font
        para: default-para
        feel: ctx-rebgui/edit/feel
    ]
]
```

or:

```
append-widget [
    my-widget: make ctx-rebgui/rebface [
        font: default-font
        para: default-para
        feel: make default-feel [
            engage: get in ctx-rebgui/edit/feel 'engage
        ]
    ]
]
```

6. Coding Standards

6.1. Init Attribute

Make use of the `init` attribute to perform any required pre-display initialization tasks. Take a look at the `tab-panel` widget for an example of its use.

6.2. Facet Order

Specify the facets of your widget in the same order as they are specified in the `face` object.

6.3. Span Attribute

Always specify span resize directives in upper-case alphabetical order (e.g. #HWXY), with span size directives preceding them in the following order: `#LVO`. The full specification sequence is thus:

```
#LVOHWXY
```

> This order is by convention only, widget code should never assume this order. For example, write:

```
all [find span #H find span #W]
```

instead of:

```
find span #HW
```

6.4. Options Block

Make use of the options block to pass and store flags and meta-data, leaving the data attribute free to store primary data. For example, it is better to have:

```
data: [1 2 3 4]
options: [blue cols 2 rows 2]
```

than:

```
data: [blue [cols 2 rows 2] [1 2 3 4]]
```

Subsequent init code can then use the following forms:

```
my-option?: find options 'my-option
my-value: select options 'my-value
```

6.5. Facet Defaults

Use the default REBOL/View facet values where possible.

6.6. Use facets instead of sub-faces

Take advantage of the fact that a face may have multiple facets (such as `text`, `image` and `effect`) specified at the same time to minimize the use of sub-faces.

6.7. Use requestor & accessor functions

Make use of the various functions.htmlRebGUI requestor & accessor functions where possible to reduce code clutter.

6.8. Reuse feel function code

While it may be tempting to copy the `false` block from the `feel/over` function into the `feel/engage/up` handler (to handle the case where the mouse is moved away while the mouse button remains pressed), call the function directly where possible.

```
if act = 'up [face/feel/over face false 0x0]
```

Although marginally less efficient, it makes changing a widget's look & feel that much easier.

6.9. Auto-size

Where practical, widgets should user-guide.html#section-3.2.5">auto-size by default with code similar to the following:

```
size: -1x-1
...
init: does [
    all [negative? size/x size/x: ...]
    all [negative? size/y size/y: ...]
]
```

6.10. Resize

The user-guide.html#section-3.1.1">span attribute will automatically adjust your widget's top-level face #offset# and/or `size`, but it is up to you to handle sub-face offset/size changes. Two common approaches are presented here. The first is to inherit certain `span` attributes when creating sub-faces in the `init` function as shown below:

```
init: has [p] [
    p: self
    insert pane make rebface [
        ...
        span: p/span
        ...
    ]
]
```

whilst the second approach is to specify a `feel/redraw` function that handles both the initial widget render (or portion thereof) and subsequent redraws as shown below:

```
feel: make default-feel [
    redraw: func [face act pos] [
        if act = 'show [
            ...
        ]
    ]
]
...
value-1: 0
value-2: 0
...
init: does [
    value-1: ...
```

```
        value-2: ...
    ]
```

The `init` function should do as much "one-off" initialization as possible so as the `redraw` function only has to deal with **changes**.

6.11. View/VID Words

Avoid using those View and VID words that are defined in the following SDK source scripts:

- %view-funcs.r
- %view-vid.r
- %view-edit.r
- %view-feel.r
- %view-images.r
- %view-styles.r
- %view-request.r

6.11.1. View

The following words **may** be used as they are part of the base View distribution (`%view-object.r`):

- as-pair
- caret-to-offset
- do-events
- hide
- hide-popup
- offset-to-caret
- show
- show-popup
- size-text
- textinfo
- unview
- view

6.11.2. gfx-funcs.r

RebGUI also uses the following functions from `%gfx-funcs.r`.

- brightness?
- confine
- edge-size?
- inside?
- outside?
- overlap?
- screen-offset?
- span?
- win-offset?
- within?

6.11.3. VID

The following words are **not** to be used as they are not part of the base View distribution:

- alert
- center-face
- choose
- clear-fields
- deflag-face
- dump-face
- dump-pane
- exists-thru?
- find-window
- flag-face
- flag-face?
- flash
- focus
- in-window?
- inform
- insert-event-func
- launch-thru
- layout
- load-image
- load-thru
- make-face
- path-thru
- read-net
- read-thru

- remove-event-func
- request
- request-color
- request-date
- request-download
- request-file
- request-list
- request-pass
- request-text
- scroll-para
- stylize
- unfocus
- vbug
- viewed?

7. Optimization Strategy

This section presents generic code optimization tips that apply to REBOL as a whole. In a broad sense, code can either be optimized for coding efficiency or execution efficiency. Where practical, RebGUI chooses the later by default.

7.1. Use Natives instead of Mezzanine functions

Replacing a mezzanine function with one or more native ones, especially within an iterated construct, is recommended. Some common examples are:

Replace	with
append block value	insert tail block value
repend block value	insert tail block reduce value
to-type	to type!
context	make object!
does body	make function! [] body
has vars body	make function! [/local vars] body
func spec body	make function! spec body
function spec vars body	make function! [spec /local vars] body
attempt []	error? try []
join	form
rejoin	reform
form	as-string
replace	change

Note that even though `as-pair` is a mezzanine function, its use is encouraged over that of `to pair! reduce`.

7.2. Avoid use of paths

Path notation, whilst simple to express, is not very efficient.

Replace	with
block/1	first block *or* pick block 1
block/1: value	poke block 1 value
insert tail effect/2	blk: last effect insert tail blk

7.3. Perform multiple assignments at once

Use:

```
v1: v2: v3: v4: v5: none
```

instead of:

```
v1: none
v2: none
v3: none
v4: none
v5: none
```

but note that doing this with series may not be appropriate. For example:

```
v1: v2: copy []
```

now refer to the **same** block.

7.4. Use 'unless instead of 'if not

Use:

```
unless flag? [...]
```

instead of:

```
if not flag? [...]
```

7.5. Use 'any and 'all instead of 'either and 'if

Use:

```
flag?: any [option-1 option-2]
all [flag? value: 0]
```

instead of:

```
flag?: either option-1 [true] [if option-2 [true]]
if flag? [
    value: 0
]
```

7.6. Take advantage of a function's return value

Use:

```
value: switch test [
    condition-1 [1]
    condition-1 [2]
]
```

instead of:

```
switch test [
    condition-1 [value: 1]
    condition-1 [value: 2]
]
```

7.7. Use reduced forms in place of serialized

Use:

```
switch type? value reduce [
    integer!    [...]
    decimal!    [...]
    binary!        [...]
]
...
switch value reduce [
    true    [...]
    false    [...]
    none    [...]
]
```

instead of:

```
switch type? value [
    #[datatype! integer!]    [...]
    #[datatype! decimal!]    [...]
    #[datatype! binary!]    [...]
]
...
switch value [
    #[true]        [...]
    #[false]    [...]
    #[none]        [...]
]
```

This is a change from what was previously stated as serialized forms are not compatible with the SDK.

7.8. Avoid Global name-space pollution

Ensure all the "local" words you use within a function are in fact declared locally. A simple way to test this is with code like the following:

```
REBOL []

query/clear system/words

do %rebgui.r

display "Test" [
    my-widget ...
]

foreach word sort query/clear system/words [
    if value? word [print word]
]

halt
```

You will see a list of RebGUI words defined in the Global context similar to the following:

```
alert
append-widget
clear-text
ctx-rebgui
display
examine
get-values
question
rebface
request-color
request-date
request-dir
request-file
request-font
request-menu
request-password
request-progress
request-spellcheck
request-ui
set-color
set-data
set-focus
set-locale
set-text
set-text-color
set-texts
set-title
set-values
splash
translate
```

Examine those not in the above list to determine if they come from your code.