R3 GUI Actors

Author: Carl Sassenrath, Saphirion AG, GC

Revision History
31-May-2013 A

Table of Contents

1. Concept

An `actor` implements a style function.

If you think of a style as an object `class`, its actors are like the `methods`, and they operate on the instance of that class, the face object. Essentially, they are functions that are reused for each instance of a style, each face.

We call them `actors` because they are are not true methods and are similar to the actor functions used in `schemes`.

.

Note

This document is is for advanced users who plan to implement their own styles and need to provide new functions for rendering, input, or other control. Casual GUI users do not need to know about actors.

2. Defining Actors

GUI styles define actors to provide functions that handle events, process input, modify attributes, and setup graphics rendering.

Within a style an `actors` block holds the actor definitions. The block is a collection of actor names followed by their function body blocks. No function creators or argument specifications are needed, the `make-style` function will create those.

Note that actor blocks are treated in the same way as the body block of a Funct or Closure construct so that all set-words used in the actor block are initialized to none. If you need to set the value of a variable defined outside the layout, you will need to specify a context for that variable or use 'set for those in the user context.

The naming convention for actors begins with `on-` followed by a verb or verb-noun that best describes the action. Examples are: `on-draw` or `on-view`. Although not strictly required, using this convention helps make GUI code more clear.

Here's an example actors block as it would look within a style definintion:

```
actors: [
        on-make: [
                ...
        ]
        on-update: [
                ...
        ]
        on-resize: [
                ...
        ]
        on-scroll: [
                ...
        ]
        on-over: [
                ...
        ]
]
```

Here is a complete style definition of `clicker`, the base-style that implements buttons.

```
clicker: [

        about: "Single-action button without text. Basis of other styles."

        tags: [internal]
```

```
	facets: [
		init-size: 28x28
		bg-color: 80.100.120
		border-color: 0.0.0.128

		pen-color: ; set by on-draw
		area-fill: ; set by on-draw
		material: 'chrome
		focus-color: guie/colors/focus
		draw-mode: 'normal
		materials: none
		face-width: none
	]

	options: [
		face-width: [integer!]
		init-size: [pair!]
		bg-color: [tuple!]
	]

	state: [
		validity: none
	]

	draw: [
		normal: [
			pen pen-color
			line-width 1
			grad-pen linear 1x1 0 (viewport-box/bottom-right/y) 90 area-fill
			box 1x1 (viewport-box/bottom-right - 2) 1
		]
		focus: [
			fill-pen focus-color
			box -1x-1 viewport-box/bottom-right 5
			pen pen-color
			line-width 1
			grad-pen linear 1x1 0 (viewport-box/bottom-right/y) 90 area-fill
			box 1x1 (viewport-box/bottom-right - 2) 1
		]
	]
	actors: [
		on-make: [
			if face/facets/face-width [
				face/facets/init-size/x: face/facets/min-size/x: face/fa
			]
		]

		on-init: [
			set-facet face 'materials make-material face get-facet face 'mat
		]

		on-draw: [
			set-material face face/state/mode
			color: get-facet face 'border-color
			if face/state/mode = 'over [color: color / 2]
			face/facets/pen-color: color
			arg ; return draw block
		]
```

```
                on-over: [; arg: offset or none
                        face/state/mode: pick [over up] face/state/over: not not arg
                        draw-face face
                ]

                on-click: [; arg: event
                        face/state/mode: arg/type
                        if 'up = face/state/mode [face/state/mode: 'over]
                        draw-face face
                        if arg/type = 'up [
                                focus face
                                do-face face
                        ]
                        true ;don't do unfocus
                ]

                on-focus: [; arg/1: TRUE for focus, FALSE for unfocus; arg/2 - forced re
                        set-facet face 'draw-mode either get arg/1 ['focus] ['normal]
                        set-facet face 'focus-color either get arg/1 [guie/colors/focus]
                        draw-face face
                ]

                on-key: [; arg: event
                        if arg/type = 'key [
                                switch arg/key [
                                        #" " [
                                                do-face face
                                        ]
                                ]
                        ]
                ]

                on-validate: [
                        face/state/validity: validate-face face
                ]
        ]
]
```

2.1. Actor Arguments

Actor arguments are fixed and standardized. The arguments are:

face	The face upon which the actor acts.
arg	A single value or block of multiple values.

When an actor is called, argument values are passed to the actor body block. Local variables can be defined using set-words (same way as in FUNCT method).

2.2. Inherited Actors

When a style is derived from another style the derived style inherits the actors of the parent style.

For example, `button` uses the `clicker` actors here:

```
button: clicker [

        about: "Single action button with text."

        tags: [action tab]

        facets: [
                init-size: 130x24
                text: "Button"
                text-style: 'button
                max-size: 260x24
                min-size: 24x24
                text-size-pad: 20x0
        ]

        options: [
                text: [string! block!]
                bg-color: [tuple!]
                init-size: [pair!]
                face-width: [integer! issue!]
        ]

        actors: [
                on-make: [
                        either face/facets/face-width = #auto [
                                face/facets/max-size:
                                face/facets/init-size: face/facets/text-size-pad + as-pa
                        ] [
                                do-actor/style face 'on-make arg 'clicker
                        ]
                ]

                on-set: [
                        if arg/1 = 'value [
                                face/facets/text: form any [arg/2 ""]
                                show-later face
                        ]
                ]
                on-get: [
                        if arg = 'value [
                                face/facets/text
                        ]
                ]
                on-draw: [
                        t: get-facet face 'text
                        ; limit-text-size modifies, so we need to copy
                        ; size is made 20px smaller to incorporate "..." (see text-size-
                        l: limit-text-size copy/deep t face/gob/size - face/facets/text-
                        set-facet face 'text-body either equal? t l [t] [join l "..."]
                        do-actor/style face 'on-draw arg 'clicker
                ]
        ]
]
```

Note that three new actors are defined here, and the `on-draw` is redefined. It actually adds further functionality before it then calls the parent style's (clicker) on-draw actor using the do-actor/style function.

2.3. Accessing Facets

For any given style a facet value may be stored in either the style object itself, or within the face instance. The location depends on whether the facet is static for all face instances, or changes for each instance.

Because you don't know and shouldn't care where it is stored, the `get-facet` function is provided to get the value and the `set-facet` function to set it.

For example, if the `on-resize` actor needs to know the size facet, it would use:

```
size: get-facet face 'size
```

If the size is found in the `face/facets` object, that will be used. Otherwise, the `style/facets` object will be used.

3. Standard Actor Names

A number of actor names are predefined for standard actions, and we recommend that you use these for their equivalent actors within your GUI styles:

Actor	Description
on-make	when face is first created to initialize special values
on-click	when mouse button clicked on face
on-drag	when dragging inside a face
on-drag-over	when dragging and over a target face
on-drop	when drag has stopped or when a file is dropped
on-focus	when we have been given or are losing focus
on-get	to fetch state values
on-set	when state values are set
on-clear	when the state values are cleared
on-key	when key has been pressed (for our focus)
on-move	when mouse has moved
on-over	when mouse passes over or away
on-reset	when reset is needed
on-resize	when size values have changed
on-update	when face contents have changed
on-draw	when system starts to draw the face (create DRAW block)
on-scroll	when scrolling is needed
on-scroll-event	when the mouse wheel is used

on-init	when face and its parent pane are ready (including initial sizing)
on-attach	when a face is attached **from** another face
on-attached	when a face is triggered in the attach reaction chain

4. Default Actors

A small number of actors are defined by default to work for all styles. They are:

Style	Description
on-resize	Recompute the area-size facet.
on-get	Return the face/state of a given name.
locate	A special actor to map an event to an offset position.
on-attached	By default sets the attached face value to the current face value.

Style definitions are allowed to override these actors with their own definition specific to their operation.

5. Calling an Actor

There will be times when one actor function will call another. This is done with the `do-actor` function. Actors are always called this way, never directly.

For example:

```
do-actor face 'on-click true
```

It should be noted that the call will be ignored if the face has no `on-click` actor.

Sometimes you can reuse actor code from another style to avoid duplicate code in multiple styles.

For example:

```
on-resize: [
        ;call the standard on-resize code from FACE style
        do-actor/style face 'on-resize arg 'face

        ;here follows specific on-resize code for this style
        ...
]
```

6. Actors for Initialization

6.1. on-make

The `on-make` actor is called when the face is created within the layout engine.

arg	none
return	none

This actor can be used to setup face facet values such as blocks of data, colors, face orientation, etc. and anything else unique to the face. If you are defining a compound style, this is a good actor to set-up any sub-faces.

It is not necessary to compute the face size within `on-make` because `on-resize` will be called later to do that.

Example that sets colors unique to the new face:

```
on-make: [
        face/facets/colors: copy [255.255.255 128.128.128]
]
```

6.2. on-init

The on-init actor is called when a new layout is created.

arg	none
return	none

When a new layout is created, the `on-init` actor will be called for each face within the layout. At this point the face objects exist and the layout has been initialized, including any special bindings.

It is also called for any trigger faces (special faces that act on `when` layout events.)

Resetting the face on initialization:

```
on-init: [
        do-actor face 'on-reset
]
```

6.3. on-attach

The `on-attach` actor is called when this face gets attached **from** another face.

arg	the face requesting attachment
return	none

Here is simple example how it works:

```
stylize [
        node: box [
                facets: [
                        text-style: 'title
                ]
                options: [
                        bg-color: [tuple!]
                        text-body: [string!]
                ]
                actors: [
```

```
                        on-attach: [
                                print ["ON-ATTACH: Node" face/name "has been requested t
                        ]
                ]
        ]
]

view [
        A: node red "Node A" attach 'B
        B: node green "Node B" attach 'A
        C: node blue "Node C" attach 'A
]
```

You should see following output in the console:

```
ON-ATTACH: Node B has been requested to attach from node A [A-&gt ;B]
        ON-ATTACH: Node A has been requested to attach from node B [B-&gt ;A]
                ON-ATTACH: Node A has been requested to attach from node C [C-&gt ;A]
```

.

Note

For easy access the face/targets field contains faces that are attached FROM the face.

7. Actors for Events

7.1. on-attached

The `on-attached` is called on every attached face contained in the targets field, and is triggered by a `do-face` function call sequence on a face that contains targets.

7.2. on-click

The `on-click` is called each time a mouse button press or release occurs.

arg	the event object for the mouse click.
return	on down, can be none or the drag object. On up, none.

This example prints the button event that occurred:

```
on-click: [
        probe arg/type
]
```

`on-click` is also the precursor to dragging (holding down a mouse button, while moving the mouse), so you can create a drag object in `on-click` using the `init-drag` function. If you return the drag object the `on-drag` actor will be invoked.

```
on-click: [
        if arg/type = 'down [
                return init-drag face arg/offset
        ]
```

```
        none ; return value
]
```

7.3. on-drag

The `on-drag` actor is called when the drag object is created or when the mouse is moving inside its target face.

arg	drag object (created earlier)
return	none

`on-drag` is usually called when an `on-click` returns a drag object, and ceases to be used when the drag object is destroyed, which happens right after `on-drop`.

Note that when a drag object exists, the `on-move` and `on-over` actors are not called.

Here is an example used by a slider that changes a value constantly during drag, updates its graphics, and calls its attached faces (if any).

```
on-drag: [; arg: drag
        ; send the event movement data to the face to allow it to update its position
        do-actor face 'on-offset arg/delta + arg/base
        ; redraw the face to show the new position
        draw-face face
        ; call any attached faces in this face's targets field
        do-face face
]
```

7.4. on-drag-over

The `on-drag-over` actor is called when the mouse is moving over a foreign face ie. a face that did not create the drag object.

arg	Block of values related to the drag [drag-object offset ???]
return	none

Examples

```
on-drag-over: [
        ; write this example
]
```

7.5. on-drop

The `on-drop` actor is called when the drag operation is released.

arg	the drag object created earlier.
return	none

At the end of a drag operation, when one of the mouse buttons is released, the `on-click` action is called with the event, then `on-drop` is called.

After `on-drop`, the drag object is automatically destroyed. The `on-drop` actor is called regardless of whether the drag operation was started over this face or a different face.

The `on-drop` actor is also called if a file is dragged and dropped from the system desktop over the face. This is a special case.

Example:

```
on-drop: [
        ; write an example here
]
```

7.6. on-focus

The `on-focus` actor is called every time the face gains focus using the `focus` or `next-focus` function or loses focus with the `unfocus` function.

arg	true for focus and none for unfocus.
return	none

Nothing is called after `on-focus`, so if the face is changing appearance, a `show-later` should be called within the actor.

This example makes the background color yello (selection variation of yellow) when the face is focused and makes it white, when the face is unfocused:

```
on-focus: [
        face/facets/bg-color: pick reduce [yello white] arg
        draw-face face
]
```

7.7. on-key

The `on-key` actor is called when a face has focus and a key is pressed.

arg	the input event
return	the same event

The `on-key` actor has a default value for all faces, to return the event argument.

Simple keyboard navigation in a street map face:

```
on-key: [
        if arg/type = 'key [;detect only "key-down" event types (use 'key-up for up even
                dx: dy: 0
                switch arg/key [
                        right [dx: 1]
                        left [dx: -1]
                        up [dy: 1]
                        down [dy: -1]
                ]
                if find arg/flags 'shift [
                        dx: dx * 3
```

```
                        dy: dy * 3
                ]
                move-map as-pair dx dy
        ]
        arg ; return same event
]
```

7.8. on-move

The `on-move` actor is called every time the mouse moves.

arg	the event value (with face-relative positions)
return	none

Important notes:

- Not required for most styles.
- Will be called often, so don't define it unless you need it.
- The `on-over` actor will be called after this actor.
- Not be called during a drag operation.

7.9. on-over

The `on-over` actor is called every time the mouse moves over or away from the face.

arg	the face relative position or none
return	none

If the face has the `all-over` value specified as true, this actor will be run continuously as long as the mouse is over the face. If `all-over` is not true, it does not report continuously, only on enter and exit.

7.10. on-resize

The `on-resize` actor is called every time the layout is resized.

arg	new size (pair! value)
return	none

Normally, you use this actor to modify the size fields of the draw block and any GOBs used within it. If your style is a compound style, the faces inside may also need to be resized.

Resizing a single face:

```
on-resize: [
        face/gob/size: arg
        set-facet face 'size arg
]
```

Resizing a compound face (of *hpanel* layout type):

```
on-resize: [
        do-actor/style face 'on-resize arg 'hpanel
        ;style specific code follows
        ....
]
```

7.11. on-scroll

The `on-scroll` actor is called from one face to scroll another face.

arg	the face object of the scroller
return	true if the attach was successful

When a scroll-bar is moved it will call this function to tell its attached target face to scroll. This is how a scroll-bar informs the other face that it is time to update. So, this actor must be defined for styles that are scrollable.

When a face is scrolled, this actor can be called directly by the `scroll` actor or by a prior attachment that occurs when a scrollable face is defined before a scrolling style.

Do not confuse this with `on-scroll-event` which takes an event to perform.

Scroll a text area face:

```
on-scroll: [; arg: scroller
        gob: sub-gob? face
        size: gob/size - gob/text/scroll
        tsize: size-text gob
        gob/text/scroll/y: min 0 arg/state/value * negate tsize/y - gob/size/y + 5
        show-later face
]
```

7.12. on-scroll-event

The `on-scroll-event` actor is called when a scroll event occurs.

arg	the event
return	the same event

Scroll events can be caused by the mouse-wheel or other such devices.

Do not confuse this actor with `on-scroll` which is used to scroll a face from another face, such as a `scroller` style.

Here's an example:

```
on-scroll-event: [
        dy: none
        switch arg/type [
                scroll-line [dy: arg/offset/y / -30]
                scroll-page [dy: negate arg/offset/y]
```

```
        ]
        if dy [bump-scroll face dy]
        none
]
```

Note that `bump-scroll` is a function specific to this style.

8. Actors for Value Control

8.1. on-get

The `on-get` actor returns values stored in the face object and is normally called by `get-face`. Normally, the default actor is all you need.

arg	the name of the variable to fetch (default is value)
return	The value fetched.

The default definition of `on-get` is:

```
on-get: [; arg: the field to get
        select face/state arg
]
```

8.2. on-set

The `on-set` actor is called by `set-face` for setting the state value of a face or any other value in the face.

arg	A block containing the name and its value.
return	none

The default name is `value`, which means set the primary value of the face.

Here's an example of `on-set` used by a clock style to set its time:

```
on-set: [
        if arg/1 = 'value [
                if date? time: arg/2 [time: time/time]
                face/state/value: time
                face/facets/clock/set-clock time
                show-later face
        ]
]
```

8.3. on-clear

The `on-clear` actor is called when a face needs to be cleared.

arg	none
return	none

This actor is called by the `clear` actor ??. It is also called when a layout that contains input fields is cleared with the `clear-layout` function.

```
on-clear: [
        clear face/facets/text-edit
        show-later face
]
```

8.4. on-reset

The `on-reset` actor is called when the face needs to be reset to a predefined initial value.

arg	none
return	none

This actor is mainly used for the `reset` ?? actor.

```
on-reset: [
        do-actor face 'on-set 0
]
```

9. Other Actors

9.1. on-draw

The `on-draw` actor allows you modify a draw block immediately before it is rendered.

arg	the current draw block for the face
return	the modified draw block

This function may be called often, every time a `draw-face` is needed. You should make this actor as efficient as possible by moving any time consuming computations to another actor such as `on-click`.

Here is an example of the `on-draw` used by buttons (from the clicker style):

```
on-draw: [
        set-material face face/state/mode
        color: get-facet face 'border-color
        if face/state/mode = 'over [color: color / 2]
        face/facets/pen-color: color
        arg ; return draw block
]
```