Red/View Graphic Engine

Table of Contents

1. Design goals

The Red/View (or just View) component is a graphic system for the Red programming language. The design goals are:

- Data-oriented, minimal API
- Virtual tree of objects as programming interface
- Realtime or deferred synchronization between the virtual tree and the display system
- Make two-way binding trivial to support
- Ability to have different backends, across different platforms
- Support OS, third-party and custom widget sets
- Low performance overhead

The virtual tree is built using face objects. Each face object maps to a graphic component on the display in a two-way binding.

2. Face object

Face objects are clones of `face!` template object. A field in face object is called a *facet*.

List of available facets:

Facet	Datatype	Mandatory?	Applicability	Description
type	word!	yes	all	Type of graphic component (see below).
offset	pair!	yes	all	Offset position from parent top-left origin.
size	pair!	yes	all	Size of the face.
text	string!	no	all	Label text displayed in the face.
image	image!	no	some	Image displayed in the face background.
color	tuple!	no	some	Background color of the face in R.G.B or R.G.B.A format.
menu	block!	no	all	Menu bar or contextual menu.

Facet	Datatype	Mandatory?	Applicability	Description
data	any-type!	no	all	Content data of the face.
enabled?	logic!	yes	all	Enable or disable input events on the face.
visible?	logic!	yes	all	Display or hide the face.
selected	integer!, pair!, object!	no	some	For lists types, index of currently selected element. For text inputs, selected text. For windows, focused face.
flags	block!, word!	no	some	List of special keywords altering the display or behavior of the face.
options	block!	no	some	Extra face properties in a [name: value] format.
parent	object!	no	all	Back-reference to parent face (if any).
pane	block!	no	some	List of child face(s) displayed inside the face.
state	block	no	all	Internal face state info *(used by View engine only)*.
rate	integer!, time!	no	all	Face's timer. Periodically fires `time` event. An `integer!` sets a frequency in Hz, a `time!` sets an interval, `none` stops it.
edge	object!	no	all	*(reserved for future use)*
para	object!	no	all	Para object reference for text positioning.
font	object!	no	all	Font object reference for setting text facet's font properties.
actors	object!	no	all	User-provided events handlers.
extra	any-type!	no	all	Optional user data attached to the face (free usage).
draw	block!	no	all	List of Draw commands to be drawn on the face.

List of globally-usable flags for `flags` facet:

Flag	Description
all-over	Send all `over` events to the face.

Other face types specific flags are documented in their respective sections.

NOTE

- Non-mandatory facets can be set to `none`.
- `offset` and `size` are specified in screen pixels.
- `offset` and `size` can sometime be set to `none` before displaying them. The View engine will take care of setting the values (like for panels in tab-panel type).
- Display order (from back to front): color, image, text, draw.

Creating a new face is achieved by cloning the `face!` object and providing **at least** a valid `type` name.

```
button: make face! [type: 'button]
```

Once a face created, the `type` field is not allowed to be changed.

2.1. Options facet

Options facet holds optional facets which are used for specific behaviors:

Option	Description
drag-on	Can be one of: `'down`, `'mid-down`, `'alt-down`, `'aux-down`. Used for enabling a drag'n drop operation.

3. Font object

Font objects are clones of `font!` template object. One font object can be referenced by one or more faces, allowing to control font properties of a group of faces from a single place.

Field	Datatype	Mandatory?	Description
name	string!	no	Valid font name installed on the OS.
size	integer!	no	Font size in points.
style	word!, block!	no	Styling mode or block of styling modes.
angle	integer!	yes	Text writing angle in degrees (default is `0`).
color	tuple!	yes	Font color in R.G.B or R.G.B.A format.
anti-alias?	logic!, word!	no	Anti-aliasing mode (active/inactive or special mode).
shadow	*(reserved)*	no	*(reserved for future use)*
state	block!	no	Internal face state info *(used by View engine only)*.
parent	block!	no	Internal back reference to parent face(s) *(used by View engine only)*.

NOTE	• Non-mandatory facets can be set to `none`. • `angle` field is not yet working properly. • All fields values should become optional in the future.

Available font styles:

- `bold`
- `italic`
- `underline`
- `strike`

Available anti-aliasing modes:

- active/inactive (`anti-alias?: yes/no`)
- ClearType mode (`anti-alias?: 'ClearType`)

4. Para object

Para objects are clones of `para!` template object. One para object can be referenced by one or more faces, allowing to control para properties of a group of faces from a single place.

Field	Datatype	Description
origin	*(reserved)*	*(reserved for future use)*
padding	*(reserved)*	*(reserved for future use)*
scroll	*(reserved)*	*(reserved for future use)*
align	word!	Control horizontal text alignment: `left`, `center`, `right`.
v-align	*(reserved)*	Control vertical text alignment: `top`, `middle`, `bottom`.
wrap?	logic!	Enable/disable text wrapping in the face(s).
parent	block!	Internal back reference to parent face(s) *(used by View engine only)*.

NOTE	• Any para fields can be set to `none`.

5. The Face tree

Faces are organized in a tree which maps to the graphic components hierarchy on the display. The tree relations are defined from:

- `pane` facet: list of one or more child face(s) in a block.
- `parent` facet: reference to parent face.

Order of face objects in a `pane` matters, it maps to the z-ordering of graphic objects (face at head of `pane` is displayed behind all other faces, the face at tail is displayed on top of all others).

The root of a face tree is a `screen` face. A `screen` face can only display `window` faces from its `pane` block.

In order for any face to be displayed on screen, it *must* be connected to a `screen` face directly (for windows) or indirectly (for other face types).

[Face tree] | *face-tree.png*

6. Face types

6.1. Base

The `base` type is the most basic face type, but also the most versatile one. By default, it will only display a background of color `128.128.128`.

Facet	Description
`type`	`'base`
`image`	An `image!` value can be specified, alpha channel is supported.
`color`	A background color can be specified, alpha channel is supported.
`text`	An optional text to be displayed inside the face.
`draw`	Transparency is fully supported for Draw primitives.

NOTE

- Full composition of following facets is supported and rendered in following order: `color`, `image`, `text`, `draw`.
- Transparency can be achieved in `color`, `image`, `text` and `draw` by specifying an alpha channel component in color tuple values: `R.G.B.A` where `A = 0` indicates full opacity and `A = 255` full transparency.

This face type should be used for any custom graphic component implementation.

6.2. Text

The `text` type is a static label to be displayed.

Facet	Description
`type`	`'text`
`text`	Label text.
`data`	Value to display as text.
`options`	Supported fields: `default`.

`data` facet is synchronized in real-time with `text` facet using the following conversion rules:

- when `text` changes, `data` is set to the `load`-ed `text` value, or `none`, or to `options/default` if defined.
- when `data` changes, `text` is set to the `form`-ed `data` value.

`options` facet accepts following properties:

- `default`: can be set to any value, it will be used by the `data` facet if converting `text` returns `none`, like for non-loadable strings.

6.3. Button

This type represents a simple button.

Facet	Description
`type`	`'button`
`text`	Button's label text.
`image`	The image will be displayed inside the button. Can be combined with a text.

Event type	Handler	Description
`click`	`on-click`	Triggered when the user clicks on the button.

6.4. Toggle

This type represents a button that retains its state after being pushed.

Facet	Description
`type`	`'toggle`
`text`	Toggle's label text.
`para`	Controls vertical and horizontal text alignment.
`data`	`true`: toggled; `false`: untoggled (default).
`image`	The image will be displayed inside the toggle. Can be combined with a text.

Event type	Handler	Description
`change`	`on-change`	Triggered when the toggle state is changed by a user action.

6.5. Check

This type represents a check box, with an optional label text, displayed on left or right side.

Facet	Description
`type`	`'check`

Facet	Description
`text`	Label text.
`para`	The `align` field controls if the text is displayed on the `left` or on the `right` side.
`data`	`true`: checked; `false`: unchecked; `none`: unchecked for 2-state check box, indeterminate for 3-state check box (default).
`flags`	Turn on tri-state check box feature (word!).

Supported flags:

- `tri-state`: enables third, indeterminate state that is represented as `none` value in `data` facet.

Event type	Handler	Description
`change`	`on-change`	Triggered when the check state is changed by a user action.

6.6. Radio

This type represents a radio button, with an optional label text, displayed on left or right side. Only one radio button per pane is allowed to be checked.

Facet	Description
`type`	`'radio`
`text`	Label text.
`para`	The `align` field controls if the text is displayed on the `left` or on the `right` side.
`data`	`true`: checked; `false`: unchecked (default).

Event type	Handler	Description
`change`	`on-change`	Triggered when the radio state is changed by a user action.

6.7. Field

This type represents a single-line input field.

Facet	Description
`type`	`'field`
`text`	Input text; read/write value.
`data`	Value to display as text.
`selected`	Selected text (pair! none!).
`options`	Supported fields: `default`.
`flags`	Turn on/off some special field features (block!).

The `selected` facet controls the text highlighting (read/write). A pair value indicates the index of first and last selected characters. A `none` value indicates that no text is selected in the field.

Supported flags:

- `no-border`: removes edge decorations made by the underlying GUI framework.
- `password`: instead of input characters, asterisks (*) are displayed.

`data` facet is synchronized in real-time with `text` facet using the following conversion rules:

- when `text` changes, `data` is set to the `load`-ed `text` value, or `none`, or to `options/default` if defined.
- when `data` changes, `text` is set to the `form`-ed `data` value.

`options` facet accepts following properties:

- `default`: can be set to any value, it will be used by the `data` facet if converting `text` returns `none`, like for non-loadable strings.

Event type	Handler	Description
`enter`	`on-enter`	Occurs each time the Enter key is pressed down in the field.
`change`	`on-change`	Occurs each time an input is made in the field.
`select`	`on-select`	Occurs each time after a text is selected using mouse or keyboard.
`key`	`on-key`	Occurs each time a key is pressed down in the field.

6.8. Area

This type represents a multi-line input field.

Facet	Description
`type`	`'area`
`text`	Input text; read/write value.
`selected`	Selected text (pair! none!).
`flags`	Turn on/off some special area features (block!).

The `selected` facet controls the text highlighting (read/write). A pair value indicates the index of first and last selected characters. A `none` value indicates that no text is selected in the area.

Supported flags:

- `no-border`: removes edge decoration made by the underlying GUI framework.

NOTE	• A vertical scroll-bar can appear if all lines of text cannot be visible in the area (might be controlled by a `flags` option in the future).

Event type	Handler	Description
`change`	`on-change`	Occurs each time an input is made in the area.
`select`	`on-select`	Occurs each time after a text is selected using mouse or keyboard.
`key`	`on-key`	Occurs each time a key is pressed down in the area.

6.9. Text-list

This type represents a vertical list of text strings, displayed in a fixed frame. A vertical scrollbar appears automatically if the content does not fit the frame.

Facet	Description
`type`	`'text-list`
`data`	List of strings to display (`block! hash!`).
`selected`	Index of selected string or none value if no selection (read/write).

Event type	Handler	Description
`select`	`on-select`	Occurs when an entry in the list is selected. `selected` facet refers to **old** selected entry index.
`change`	`on-change`	Occurs after a `select` event. `selected` facet refers to the **new** selected entry index.

NOTE

- number of visible items cannot yet be defined by user.

6.10. Drop-list

This type represents a vertical list of text strings, displayed in a foldable frame. A vertical scrollbar appears automatically if the content does not fit the frame.

Facet	Description
`type`	`'drop-list`
`data`	List of strings to display (`block! hash!`).
`selected`	Index of selected string or `none` value if no selection (read/write).

The `data` facet accepts arbitrary values, but only string values will be added to the list and displayed. Extra values of non-string datatype can be used to create associative arrays, using strings as keys. The `selected` facet is a 1-based integer index indicating the position of the selected string in the list, and not in the `data` facet.

Supported flags:

NOT YET IMPLEMENTED	• `scrollable`: Manually enable a vertical scroll-bar.

Event type	Handler	Description
`select`	`on-select`	Occurs when an entry in the list is selected. `selected` facet refers to **old** selected entry index.
`change`	`on-change`	Occurs after a `select` event. `selected` facet refers to the **new** selected entry index.

NOTE	• number of visible items cannot yet be defined by user.

6.11. Drop-down

This type represents an edit field with a vertical list of text strings displayed in a foldable frame. A vertical scrollbar appears automatically if the content does not fit the frame.

Facet	Description
`type`	`'drop-down`
`data`	List of strings to display (`block! hash!`).
`selected`	Index of selected string or `none` value if no selection (read/write).

The `data` facet accepts arbitrary values, but only string values will be added to the list and displayed. Extra values of non-string datatype can be used to create associative arrays, using strings as keys. The `selected` facet is a 1-based integer index indicating the position of the selected string in the list, and not in the `data` facet.

Supported flags:

NOT YET IMPLEMENTED	• `scrollable`: Manually enable a vertical scroll-bar.

Event type	Handler	Description
`select`	`on-select`	Occurs when an entry in the list is selected. `selected` facet refers to **old** selected entry index.
`change`	`on-change`	Occurs after a `select` event. `selected` facet refers to the **new** selected entry index.

NOTE	• number of visible items cannot yet be defined by user.

6.12. Calendar

This type represents a monthly Gregorian calendar in the range from 1-Jan-1601 to 31-Dec-9999.

Facet	Description
`type`	`'calendar`

Facet	Description
`data`	`date!` value that represents selected day.

Event type	Handler	Description
`change`	`on-change`	Occurs when a date in the calendar is selected.

NOTE

- By default, `data` facet is initialized to "today" date.
- `date!` value below or above specificed calendar boundaries selects minimum or maximum supported date, respectively.

6.13. Progress

This type represents a horizontal or vertical progress bar.

Facet	Description
`type`	`'progress`
`data`	Value representing the progression (`percent!` or `float!` value).

NOTE

- if a float value is used for `data`, it needs to be between 0.0 and 1.0.

6.14. Slider

This type represents a cursor which can be moved along a horizontal or vertical axis.

Facet	Description
`type`	`'slider`
`data`	Value representing the cursor position (`percent!` or `float!` value).

NOTE

- if a float value is used for `data`, it needs to be between 0.0 and 1.0.

6.15. Camera

This type is used to display a video camera feed.

Facet	Description
`type`	`'camera`
`data`	List of camera(s) name(s) as a block of strings.
`selected`	Select the camera to display from `data` list, using an integer index. If set to `none`, the camera feed is disabled.

NOTE

- The `data` facet is initially set to `none`. The list of cameras is fetched during the first call to `show` on the camera face.
- It is possible to capture the content of a camera face using `to-image` on the face.

6.16. Panel

A panel is a container for other faces.

Facet	Description
`type`	`'panel`
`pane`	Block of children faces. Order in block defines z-order on display.

NOTE

- Children `offset` coordinates are relative to parent's panel top-left corner.
- Children faces are clipped into the panel frame.

6.17. Tab-panel

A tab-panel is a list of panels where only one can be visible at a given time. A list of panels names is displayed as "tabs", and used to switch between the panels.

Facet	Description
`type`	`'tab-panel`
`data`	Block of tabs names (string values).
`pane`	List of panels corresponding to tabs list (`block!`).
`selected`	Index of selected panel or none value (`integer!`) (read/write).

Event type	Handler	Description
`change`	on-change	Occurs when user selects a new tab. `event/picked` holds the index of the newly selected tab. `selected` property is updated just after this event.

NOTE

- Both `data` and `pane` facets need to be filled in order for the tab-panel to be displayed properly.
- If `pane` contains more panels than specified tabs, they will be ignored.
- When adding/removing a tab, the corresponding panel needs to be added/removed too to/from `pane` list.

6.18. Window

Represents a window displayed on the OS desktop.

Facet	Description
type	'window
text	Title of the window (string!).
offset	Offset from top-left corner of the desktop screen, not counting the window's frame decorations. (pair!)
size	Size of the window, not counting the window's frame decorations. (pair!)
flags	Turn on/off some special window features (block!).
menu	Displays a menu bar in the window (block!).
pane	List of faces to display inside the window (block!).
selected	Select the face which will get the focus (object!).

Supported flags:

- modal: makes the window modal, disabling all previously opened windows.
- resize: enable window resizing (default is fixed size, not resizeable).
- no-title: do not display a window's title text.
- no-border: remove window's frame decorations.
- no-min: remove minimize button from window's drag bar.
- no-max: remove maximize button from window's drag bar.
- no-buttons: remove all buttons from window's drag bar.
- popup: alternative smaller frame decoration (Windows only).

NOTE	• Using the popup keyword at the beginning of the menu specification block will force a contextual menu in the window, instead of a menu bar by default.

6.19. Screen

Represents a graphic display unit connected to the computer (usually a monitor).

Facet	Description
type	'screen
size	Size of the screen display in pixels. Set by the View engine when started (pair!).
pane	List of windows to display on the screen (block!).

All window faces which are displayed need to be children of a screen face.

6.20. Group-box

A group-box is a container for other faces, with a visible frame around it. *This is a temporary style*

which will be removed once we have the support for `edge` *facet.*

Facet	Description
`type`	`'group-box`
`pane`	Block of children faces. Order in block defines z-order on display.

NOTE

- Children `offset` coordinates are relative to group-box's top-left corner.
- Children faces are clipped into the group-box frame.

7. Face life cycle

1. Create a face object from the `face!` prototype.
2. Insert the face object in a face tree connected to a screen face.
3. Use `show` to render the face object on screen.
   a. system resources are allocated at this point
   b. `face/state` block is set.
4. Remove the face from the pane to remove it from the display.
5. The garbage collector will take care of releasing the system resources associated when the face is not referenced anymore.

NOTE

- A `free` function might be provided for manual control of system resources freeing for resources hungry applications.

8. SHOW function

Syntax

```
show <face>

<face>: clone of face! object or block of face objects or names (using word! values).
```

Description

This function is used to update a face or a list of faces on screen. Only a face which is referenced in a face tree connected to a screen face can be properly rendered on screen. When called the first time, system resources will be allocated, the `state` facet will be set and the graphic component will be displayed on screen. Subsequent calls will reflect on screen any change made to the face object. If `pane` facet is defined, `show` will also apply to the children faces recursively.

State facet

The following information is provided only for reference, in normal operation, the `state` *facet should*

be left untouched by the user. However, it can be accessed if OS API are called directly by user or if View engine behavior has to be modified.

Position/Field	Description
1 (handle)	OS-specific handle for the graphic object (`integer!`).
2 (changes)	Bit flags array marking which facet has been changed since last call to `show` (`integer!`).
3 (deferred)	List of deferred changes since last call to `show`; when realtime updates are turned off (`block! none!`).
4 (drag-offset)	Stores the starting mouse cursor offset position when entering face dragging mode (`pair! none!`).

NOTE

- After a call to `show`, `changes` field is reset to 0 and `deferred` field block is cleared.
- A `handle!` datatype will be used in the future for opaque OS handles.

9. Realtime vs deferred updating

The View engine has two different modes for updating the display after changes are done to the face tree:

- Realtime updating: any change to a face is immediately rendered on screen.
- Deferred updating: all changes to a face are not propagated on screen, until `show` is called on the face, or on the parent face.

The switching between those modes is controlled by the `system/view/auto-sync?` word: if set to `yes`, the realtime updating mode is on (default mode), if set to `no`, View engine will defer all updates.

The motivations for realtime updating by default are:

- Simpler and shorter source code, no need to call `show` after any face change.
- Less learning overhead for beginners.
- Good enough for simple or prototype apps.
- Simplifies experimentation from console.

Deferred mode updates many changes at the same time on screen in order to avoid glitches or when best performance is the goal.

NOTE

- This is a big difference with the Rebol/View engine which only has deferred mode support.

10. Two-way binding

Face objects rely on the Red ownership system to bind the object with the series used in facets, so

that any change in one of the facet (even a deep change) is detected by the face object and processed according to the current synchronization mode (realtime or deferred).

On the other side, changes made to the rendered graphic objects are reflected instantly in the corresponding facets. For example, typing in a `field` face will reflect the input in the `text` facet in live mode.

This two-way binding simplifies the interaction with the graphic objects for the programmer, without the need of any specific API. Modifying the facets using the series actions is enough.

Example:

```
view [
    list: text-list data ["John" "Bob" "Alice"]
    button "Add" [append list/data "Sue"]
    button "Change" [lowercase pick list/data list/selected]
]
```

11. Events

11.1. Event names

Name	Input type	Cause
down	mouse	Left mouse button pressed.
up	mouse	Left mouse button released.
mid-down	mouse	Middle mouse button pressed.
mid-up	mouse	Middle mouse button released.
alt-down	mouse	Right mouse button pressed.
alt-up	mouse	Right mouse button released.
aux-down	mouse	Auxiliary mouse button pressed.
aux-up	mouse	Auxiliary mouse button released.
drag-start	mouse	A face dragging starts.
drag	mouse	A face is being dragged.
drop	mouse	A dragged face has been dropped.
click	mouse	Left mouse click (button widgets only).
dbl-click	mouse	Left mouse double-click.
over	mouse	Mouse cursor passing over a face. This event is produced once when the mouse enters the face and once when it exits. If `flags` facet contains **all-over** flag, then all intermediary events are produced too.

Name	Input type	Cause
move	mouse	A window has moved.
resize	mouse	A window has been resized.
moving	mouse	A window is being moved.
resizing	mouse	A window is being resized.
wheel	mouse	The mouse wheel is being moved.
zoom	touch	A zooming gesture (pinching) has been recognized.
pan	touch	A panning gesture (sweeping) has been recognized.
rotate	touch	A rotating gesture has been recognized.
two-tap	touch	A double tapping gesture has been recognized.
press-tap	touch	A press-and-tap gesture has been recognized.
key-down	keyboard	A key is pressed down.
key	keyboard	A character was input or a special key has been pressed (except control; shift and menu keys).
key-up	keyboard	A pressed key is released.
enter	keyboard	Enter key is pressed down.
focus	any	A face just got the focus.
unfocus	any	A face just lost the focus.
select	any	A selection is made in a face with multiple choices.
change	any	A change occurred in a face accepting user inputs (text input or selection in a list).
menu	any	A menu entry is picked.
close	any	A window is closing.
time	timer	The delay set by face's `rate` facet expired.

NOTE

- touch events are not available for Windows XP.
- One or more `moving` events always precedes a `move` one.
- One or more `resizing` events always precedes a `resize` one.

11.2. Event! datatype

An event value is an opaque object holding all the information about a given event. You access the event fields using path notation.

Field	Returned value
`type`	Event type (`word!`).
`face`	Face object where the event occurred (`object!`).

Field	Returned value
window	Window face where the event occured (object!).
offset	Offset of mouse cursor relative to the face object when the event occurred (pair!). For gestures events, returns the center point coordinates.
key	Key pressed (char! word!).
picked	New item selected in a face (integer! percent!). For a mouse down event on a text-list, it returns the item index underneath the mouse or none. For wheel event, it returns the number of rotation steps. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user. For menu event, it returns the corresponding menu ID (word!). For zooming gesture, it returns a percent value representing the relative increase/decrease. For other gestures, its value is system-dependent for now (Windows: ullArguments, field from GESTUREINFO).
flags	Returns a list of one or more flags (see list below) (block!).
away?	Returns true if the mouse cursor exits the face boundaries (logic!). Applies only if over event is active.
down?	Returns true if the mouse left button was pressed (logic!).
mid-down?	Returns true if the mouse middle button was pressed (logic!).
alt-down?	Returns true if the mouse right button was pressed (logic!).
ctrl?	Returns true if the CTRL key was pressed (logic!).
shift?	Returns true if the SHIFT key was pressed (logic!).

List of possible flags from event/flags:

- away
- down
- mid-down
- alt-down
- aux-down
- control
- shift

NOTE	
	• All fields (except type) are read-only. Setting type is only used internally by the View engine.

Here is the list of special keys returned as words by event/key:

- page-up
- page-down
- end

- `home`
- `left`
- `up`
- `right`
- `down`
- `insert`
- `delete`
- `F1`
- `F2`
- `F3`
- `F4`
- `F5`
- `F6`
- `F7`
- `F8`
- `F9`
- `F10`
- `F11`
- `F12`

The following extra key names can be returned by `event/key` only for `key-down` and `key-up` messages:

- `left-control`
- `right-control`
- `left-shift`
- `right-shift`
- `left-menu`
- `right-menu`

11.3. Actors

Actors are handler functions for View events. They are defined in an free-form object (no prototype provided) referred by `actors` facet. All actors have the same specification block.

Syntax

```
on-<event>: func [face [object!] event [event!]]

<event> : any valid event name (from above table)
face    : face object which receives the event
event   : event value.
```

In addition to the GUI events, it is possible to define an `on-create` actor which will be called when the face is shown for the first time, just before system resources are allocated for it. Unlike other actors, `on-create` has only one argument, `face`.

Return value

```
'stop : exits the event loop.
'done : stops the event from flowing to the next face.
```

Other returned values have no effect.

11.4. Event flow

Events are usually generated at a specific screen position and assigned to the closest front face. However, the event is travelling from one face to another in the ancestors hierarchy in two directions commonly known as:

- event **capturing**: event goes from window face down to the front face where the event originated. For each face, a `detect` event is generated and the corresponding handler called if provided.
- event **bubbling**: event goes from face to parent window. For each face, the local event handler is called.

[Event flow] | *event-flow.png*

Typical event flow path:

1. A click event is generated on the button, global handlers are processed (see next section).
2. Event capturing stage starts:
   a. The window gets the event first, its `on-detect` handler gets called.
   b. The panel gets the event next. Panel's `on-detect` handler gets called.
   c. The button gets the event last. Button's `on-detect` gets called.
3. Event bubbling stage starts:
   a. The button gets the event first, its `on-click` handler gets called.
   b. The panel gets the event next. Panel's `on-click` handler gets called.
   c. The window gets the event last, its `on-click` handler gets called.

NOTE

- Event cancellation is achieved by returning `'done` word from any event handler.
- Event capturing is not enabled by default for performance reasons. Set `system/view/capturing?: yes` to enable it.

11.5. Global event handlers

Before entering the event flow path, specific pre-processing can be achieved using the so-called "global event handlers". Following API is provided for adding and removing them.

11.5.1. insert-event-func

Syntax

```
insert-event-func <handler>

<handler> : a handler function or block of code for pre-processing event(s).

Handler's function specification: func [face [object!] event [event!]]
```

Return value

```
The newly added handler function (`function!`).
```

Description

Installs a global handler function, which can pre-process events before they reach the face handlers. All global handlers are called on each event, so the handler's body code needs to be optimized for speed and memory usage. If a block is provided as argument, it will be converted to a function using the `function` constructor.

The return value of the handler function:

- `none` : the event can be processed by other handlers (`none!`).
- `'done` : other global handlers are skipped but event is propagated to child faces (`word!`).
- `'stop` : exit the event loop (`word!`).

A reference to the handler function is returned and should be saved if it needs to be removed later.

11.5.2. remove-event-func

Syntax

```
remove-event-func <handler>

<handler> : a previously installed event handler function.
```

Description

Disables a previously installed global event handler by removing it from the internal list.

12. System/view object

Word	Description
`screens`	List of screen faces representing connected displays.
`event-port`	*reserved for future use*
`metrics`	*reserved for future use*
`platform`	View engine low-level platform code (includes backend code).
`VID`	VID processing code.
`handlers`	List of global event handlers
`reactors`	Internal associative table for reactive faces and their action blocks.
`evt-names`	Internal table for event to actor names conversion.
`init`	View engine initialization function, can be called by user if required.
`awake`	Main high-level events entry point function.
`capturing?`	`yes` = enables event capturing stage and `detect` events generation (default to `no`).
`auto-sync?`	`yes` = realtime faces updates (default), `no` = deferred faces updates.
`debug?`	`yes` = output verbose logs of View internal events (default to `no`).
`silent?`	`yes` = do not report VID or Draw dialects processing errors (default to `no`).

13. Including View component

View component is not included by default on **compiling**. To include it, the main Red script have to declare the dependency in the header using the `Needs` field:

```
Red [
    Needs: 'View
]
```

NOTE	Using consoles auto-generated by `red` binary will include the View component on platforms where it is available, `Needs` header field is therefore not required in user scripts run from those consoles.

14. Extra functions

Function	Description
view	Render on screen a window from a face tree or a block of VID code. Enters an event loop unless `/no-wait` **refinement** is used.
unview	Destroy one or more window(s).
layout	Convert a block of VID code into a face tree.
center-face	Center a face relatively to its parent.
dump-face	Output a compact description of a face tree structure (debugging purpose).
do-actor	Evaluate a face actor manually.
do-events	Launch an event loop (optionally just process pending events and return).
draw	Render a Draw dialect block onto an image.
to-image	Convert any rendered face to an image.
set-focus	Sets focus on a specific face.
size-text	Measure the size in pixels of a text in a face (taking the selected font into account).

To be added:

- `menu` facet specification
- `image!` datatype description