Visual Interface Dialect

Table of Contents

1. Overview

VID stands for Visual Interface Dialect. Its purpose is to provide a simple dialect (DSL) for describing graphic user interfaces on top of the Red View engine.

VID allows you to specify each graphic component to display, giving the choice of different layout methods:

- horizontal or vertical flowing
- grid positioning

- absolute positioning

VID will create a container face for you automatically, to hold the faces description you provide. By default, the container face is of type `window`.

VID code is processed by the `layout` function (which is internally called by `view` function). The VID code is then compiled to a tree of faces, suitable for direct display.

NOTE	Use `help view` and `help layout` from Red's console to see how to process a VID block.

2. Code structure

Typical VID block of code has the following structure:

```
[
    <container settings>
    <layout description>
]
```

- **container settings**: settings which affect the container object (could be a panel or a window).
- **layout description**: layout positioning commands, style definitions and face descriptions.

NOTE	All sections are optional, there is no mandatory content that must be provided in a VID block.

3. Container settings

NOTE	`react` keyword can also be used at the container settings level in addition to face options level, see its description here.

3.1. title

Syntax

```
title <text>

<text> : title text (string!).
```

Description

Sets the title text of the container face.

3.2. size

Syntax

```
size <value>

<value> : width and height in pixels (pair!).
```

Description

Sets the size of the container face. If the size is not explicitly provided, the container's size is automatically calculated to fit its content.

3.3. backdrop

Syntax

```
backdrop <color>

<color> : name or value of a color (word! tuple! issue!).
```

Description

Sets the background color of the container face.

3.4. Actors definition

Container's actors can be also be defined in this code area. See Actors section for defining actors.

4. Layout

By default, VID places the faces in the container face according to simple rules:

- direction can be horizontal or vertical
- faces are positioned after each other in the current direction using the current spacing

Defaults:

- origin: `10x10`
- space: `10x10`
- direction: `across`
- alignment: `top`

This is how faces are laid out in `across` mode:

[across] | *across.png*

This is how faces are laid out in `below` mode (using default `left` alignment):

[below] | *below.png*

4.1. across

Syntax

```
across <alignment>

<alignment> : (optional) possible values: top | middle | bottom.
```

Description

Sets the layout direction to horizontal, from left to right. An alignment modifier can be optionally provided to change the default (`top`) alignment of faces in the row.

4.2. below

Syntax

```
below <alignment>

<alignment> : (optional) possible values: left | center | right.
```

Description

Sets the layout direction to vertical, from top to bottom. An alignment modifier can be optionally provided to change the default (`left`) alignment of faces in the column.

4.3. return

Syntax

```
return <alignment>

<alignment> : (optional) possible values: left | center | right | top | middle |
bottom.
```

Description

Moves the position to the next row or column of faces, depending on the current layout direction. An alignment modifier can be optionally provided to change the current alignment of faces in the row or column.

4.4. space

Syntax

```
space <offset>

<offset> : new spacing value (pair!).
```

Description

Sets the new spacing offset which will be used for placement of following faces.

4.5. origin

Syntax

```
origin <offset>

<offset> : new origin value (pair!).
```

Description

Sets the new origin position, relative to container face.

4.6. at

Syntax

```
at <offset>
at <expr>

<offset> : position of next face (pair!).
<expr>   : Red expression returning a pair! value used as position.
```

Description

Places the next face at an absolute position. This positioning mode only affects the next following face, and does not change the layout flow position. So, the following faces, after the next one, will be placed again in the continuity of the previous ones in the layout flow.

4.7. pad

Syntax

```
pad <offset>

<offset> : relative offset (pair!).
```

Description

Modifies the layout current position by a relative offset. All the following faces on the same row (or column) are affected.

4.8. do

Syntax

```
do <body>

<body> : code to evaluate (block!).
```

Description

Evaluates a block of regular Red code, for last-minute initialization needs. The body block is bound to the container face (window or panel), so direct access to container's facet is possible. The container itself can be referred to using the `self` keyword.

5. Additional styles

View engine provides many built-in widgets, VID dialect extends them by defining additional commonly used styles, with associated keywords. They can be used with the same options as their underlying face type. They can also be re-styled freely using `style` command.

5.1. h1

The `H1` style is a `text` type with a font size set to 32.

5.2. h2

The `H2` style is a `text` type with a font size set to 26.

5.3. h3

The `H3` style is a `text` type with a font size set to 22.

5.4. h4

The `H4` style is a `text` type with a font size set to 17.

5.5. h5

The `H5` style is a `text` type with a font size set to 13.

5.6. box

The `box` style is a `base` type with a default transparent color.

5.7. image

The `image` style is a `base` type of default size 100x100. It expects an `image!` option, if none is provided, an empty image with white background color, and of same size as the face, is provided.

6. Faces definition

A face can be inserted in the layout, at the current position, simply by using the name of an existing face type or one of the available styles.

Syntax

```
<name>: <type> <options>

<name>    : optional name for the new component (set-word!).
<type>    : a valid face type or style name (word!).
<options> : optional list of options.
```

If a name is provided, the word will reference the `face!` object created by VID from the face description.

Default values are provided for each face type or style, so a new face can be used without having to specify any option. When options are required, the following sections are describing the different types of accepted options:

- Keywords
- Datatypes
- Actors

All options can be specified in arbitrary order, following the face or style name. A new face name or a layout keyword marks the end of the options list for a given face.

> **NOTE** `window` cannot be used as a face type.

6.1. Keywords

6.1.1. left

Syntax

```
left
```

Description

Aligns the face's text to left side.

6.1.2. center

Syntax

```
center
```

Description

Centers the face's text.

6.1.3. right

Syntax

```
right
```

Description

Aligns the face's text to right side.

6.1.4. top

Syntax

```
top
```

Description

Vertically align the face's text to `top`.

6.1.5. middle

Syntax

```
middle
```

Description

Vertically align the face's text to `middle`.

6.1.6. bottom

Syntax

```
bottom
```

Description

Vertically align the face's text to `bottom`.

6.1.7. bold

Syntax

```
bold
```

Description

Sets the face's text style to `bold`.

6.1.8. italic

Syntax

```
italic
```

Description

Sets the face's text style to `italic`.

6.1.9. underline

Syntax

```
underline
```

Description

Sets the face's text style to `underline`.

6.1.10. extra

Syntax

```
extra <expr>

<expr> : any value or Red expression (any-type!).
```

Description

Sets the face's `extra` facet to a value.

6.1.11. data

Syntax

```
data <list>
data <expr>

<list> : literal list of items (block!).
<expr> : Red expression returning a list as a block! value.
```

Description

Sets the face's `data` facet to a list of values. Format of the list depends on the face type requirements.

6.1.12. draw

Syntax

```
draw <commands>
draw <expr>

<commands> : literal commands list (block!).
<expr>     : Red expression returning a block! of commands.
```

Description

Sets the face's `draw` facet to a list of Draw dialect commands. See Draw dialect documentation for valid commands.

6.1.13. font

Syntax

```
font <spec>

<spec> : a valid font specification (block! object! word!).
```

Description

Sets the face's `font` facet to a new `font!` object. Font! object is described here.

NOTE	It's possible to use `font` along with other font-related settings, VID will merge them together, giving priority to the last one specified.

6.1.14. para

Syntax

```
para <spec>

<spec> : a valid para specification (block! object! word!).
```

Description

Sets the face's `para` facet to a new `para!` object. Para! object is described here.

NOTE	It possible to use `para` along with other para-related settings, VID will merge them together, giving priority to the last one specified.

6.1.15. wrap

Syntax

```
wrap
```

Description

Wrap the face's text when displaying.

6.1.16. no-wrap

Syntax

```
no-wrap
```

Description

Avoid wrapping the face's text when displaying.

6.1.17. font-size

Syntax

```
font-size <pt>

<pt> : font size in points (integer! word!).
```

Description

Sets the current font size for the face's text.

6.1.18. font-color

Syntax

```
font-color <value>

<value> : color of the font (tuple! word! issue!).
```

Description

Sets the current font color for the face's text.

6.1.19. font-name

Syntax

```
font-name <name>

<name> : valid name of an available font (string! word!).
```

Description

Sets the current font name for the face's text.

6.1.20. react

This keyword can be used both as a face option or as a global keyword. Arbitrary number of `react` instances can be used.

Syntax

```
react [<body>]
react later [<body>]

<body> : regular Red code (block!).
```

Description

Creates a new reactor from the body block. When `react` is used as a face option, the body can refer to the current face using `face` word. When `react` is used globally, target faces need to be accessed using a name. The optional `later` keyword skips the first reaction happening immediately after the `body` block is processed.

NOTE	Reactors are part of the reactive programming support in View, which documentation is pending. In a nutshell, the body block can describe one or more relations between faces properties using paths. Set-path setting a face property are processed as **target** of the reactor (the face to update), while path accessing a face property are processed as **source** of the reactor (a change on a source triggers a refresh of the reactor's code).

6.1.21. loose

Syntax

```
loose
```

Description

Enables dragging of the face using the left mouse button.

6.1.22. all-over

Syntax

```
all-over
```

Description

Sets the face `all-over` flag, allowing all mouse `over` events to be received.

6.1.23. hidden

Syntax

```
hidden
```

Description

Makes the face invisible by default.

6.1.24. disabled

Syntax

```
disabled
```

Description

Disables the face by default (the face will not process any event until it is enabled).

6.1.25. password

Syntax

```
password
```

Description

Hides user's input in a text field.

6.1.26. tri-state

Syntax

```
tri-state
```

Description

Enables 3-state mode of a check box.

6.1.27. select

Syntax

```
select <index>

<index> : index of selected item (integer!).
```

Description

Sets the `selected` facet of the current face. Used mostly for lists to indicate which item is pre-selected.

6.1.28. focus

Syntax

```
focus
```

Description

Gives the focus to the current face when the window is displayed for the first time. Only one face can have the focus. If several `focus` options are used on different faces, only the last one will get the focus.

6.1.29. hint

Syntax

```
hint <message>

<message> : hint text (string!).
```

Description

Provides a hint message inside `field` faces, when the field's content is empty. That text disappears when any new content is provided (user action or setting the `face/text` facet).

6.1.30. rate

Syntax

```
rate <value>
rate <value> now

<value>: duration or frequency (integer! time!).
```

Description

Sets a timer for the face from a duration (time!) or a frequency (integer!). At each timer's tick, a `time` event will be generated for that face. If `now` option is used, a first time event is generated immediately.

6.1.31. default

Syntax

```
default <value>

<value>: a default value for `data` facet (any-type!).
```

Description

Defines a default value for `data` facet when the conversion of `text` facet returns `none`. That default value is stored in `options` facet, as a key/value pair.

> **NOTE** currently used only by `text` and `field` face types.

6.1.32. with

Syntax

```
with <body>

<body>: a block of Red code bound to the current face (block!).
```

Description

Evaluates a block of Red code bound to the currently defined face. Allows directly setting the face fields, overriding other VID options.

6.2. Datatypes

In addition to keywords, it is allowed to pass settings to faces using literal values of following types:

Datatype	Purpose
integer!	Specifies the width of the face. For panels, indicates the number of row or columns for the layout, depending on the current direction.
pair!	Specifies the width and height of the face.
tuple!	Specifies the color of the face's background (where applicable).
issue!	Specifies the color of the face's background using hex notation (#rgb, #rrggbb, #rrggbbaa).
string!	Specifies the text to be displayed by the face.
date!	Sets the `data` facet (useful for `calendar` type).
percent!	Sets the `data` facet (useful for `progress` and `slider` types).
logic!	Sets the `data` facet (useful for `toggle`, `check` and `radio` types).
image!	Sets the image to be displayed as face's background (where applicable).
url!	Loads the resource pointed to by the URL, then process the resource according to its loaded type.

Datatype	Purpose
block!	Sets the action for the default event of the face. For panels, specifies their content.
get-word!	Uses an existing function as actor.
char!	*(reserved for future use).*

6.3. Actors

An actor can be hooked to a face by specifying a literal block value or an actor name followed by a block value.

Syntax

```
<actor>
on-<event> <actor>

<actor> : actor's body block or actor reference (block! get-word!).
<event> : valid event name (word!).
```

Description

It is possible to specify actors in a simplified way by providing just the body block of the actor, the spec block being implicit. The actor function gets constructed then and added to the face's `actor` facet. Several actors can be specified that way.

The created actor function full specification is:

```
func [face [object!] event [event! none!]][...body...]
```

The valid list of event names can be found here.

When a block or a get-word is passed without any actor name prefix, the default actor for the face type is created according to the definitions here.

7. Panels

It is possible to define child panels for grouping faces together, and eventually applying specific styles. The size of the new panel, if not specified explicitly, is automatically calculated to fit its content.

The panel-class face types from View are supported in VID with a specific syntax:

7.1. panel

Syntax

```
panel <options> [<content>]

<options> : optional list of settings for the panel.
<content> : panel's VID content description (block!).
```

Description

Constructs a child panel inside the current container, where the content is another VID block. In addition to other face options, an integer divider option can be provided, setting a grid-mode layout:

- if the direction is `across`, divider represents number of columns.
- if the direction is `below`, divider represents number of rows.

7.2. group-box

Syntax

```
group-box <divider> <options> [<body>]

<divider> : optional number of row or columns (integer!).
<options> : optional list of settings for the panel.
<body>    : panel's VID content description (block!).
```

Description

Constructs a child group-box panel inside the current container, where the content is another VID block. A divider argument can be provided, setting a grid-mode layout:

- if the direction is `across`, divider represents number of columns.
- if the direction is `below`, divider represents number of rows.

> **NOTE** Providing a `string!` value as option will set the group-box title text.

7.3. tab-panel

Syntax

```
tab-panel <options> [<name> <body>...]

<options> : optional list of settings for the panel.
<name>    : a tab's title (string!).
<body>    : a tab's content as VID description (block!).
```

Description

Constructs a tab-panel inside the current container. The spec block must contain a pair of name and content description for each tab. Each tab's content body is a new child panel face, acting as any other panels.

8. Styling

8.1. style

Syntax

```
style <new> <old> <options>

<new>     : name of new style (set-word!).
<old>     : name of old style (word!).
<options> : optional list of settings for the new style.
```

Description

Sets a new style in the current panel. The new style can be created from existing face types or from other styles. The new style is valid only in the current panel and child panels.

Styles can be cascaded from parent panels to child panels, so that the same style name can be redefined or extended in child panels without affecting the definitions in parent panels.

9. Cross-Platform GUI Metrics

In order to cope with different UI guidelines across GUI platforms, VID includes a rule-oriented GUI rewriting engine that is capable of modifying a face tree dynamically according to pre-set rules. It is integrated as a last stage in VID processing.

Windows rules:

- color-backgrounds: color the background of some colorless faces to match their parent's color
- color-tabpanel-children: Like color-backgrounds, but tab-panel specific
- OK-Cancel: buttons ordering rule, puts Cancel/Delete/Remove buttons last

macOS rules:

- adjust-buttons: use standard button sub-classes when buttons are narrow enough
- capitalize: capitalize widget text according to macOS guidelines
- Cancel-OK: buttons ordering rule, puts Ok/Save/Apply buttons last

---

A simple example, which leverages the buttons ordering and capitalization rules:

```
view [
    text "Name" right 50 field return
    text "Age"  right 50 field return
    button "ok" button "cancel"
]
```

Notice the button text and ordering on the macOS and Windows generated forms.

[mac]

[windows]

The GUI rules have ensured that:

- The buttons are ordered according to each platform's guidelines, "Ok" last on macOS, "Cancel" last on Windows.
- The button's labels are properly capitalized on macOS.

You can disable GUI-rules by setting `system/view/VID/GUI-rules/active?` to no.

```
system/view/VID/GUI-rules/active?: no
```

You can also remove rules selectively, by modifying the content of the following lists:

```
system/view/VID/GUI-rules/OS/Windows
== [
    color-backgrounds
    color-tabpanel-children
    OK-Cancel
]
```

```
system/view/VID/GUI-rules/OS/macOS
== [
    adjust-buttons
    capitalize
    Cancel-OK
]
```

This allows you total control where needed, but also helps you conform to UI guidelines with no effort.