The REBOL Documentation Project

-- FR - Documentation REBOL - Articles Techniques --

Articles Techniques

Utilisation des drapeaux (flags) avec les styles View

DideC

Première publication : 28 avril 2005, et mis en ligne le vendredi 29 avril 2005

Résumé :

Parmi tous les attributs des objets *faces* de View, il en est un qui sert à beaucoup de choses et pour lequel il n'y a pourtant aucune documentation officielle : l'attribut **flags**. Dans cet article, découvrez tous ses usages et comment l'utiliser à vos propres fins.

Parmi tous les attributs des objets *faces* de View, il en est un qui sert à beaucoup de choses et pour lequel il n'y a pourtant aucune documentation officielle : l'attribut **flags**. Dans cet article, découvrez tous ses usages et comment l'utiliser à vos propres fins.

Version : 1.0.2

Date : 28 avril 2005

Historique

Date	Version	Commentaires	Auteur	Email
10-03-2005	1.0.0	Première diffusion	Didier Cadieu	didec-at-wanadoo-dot-fr
23-04-2005	1.0.1	Corrections mineures	Didier Cadieu	didec-at-wanadoo-dot-fr
28-04-2005	1.0.2	Grammaire	Didier Cadieu	didec-at-wanadoo-dot-fr

Introduction

Les styles de View permettent de créer des interfaces graphiques rapidement et simplement grâce au dialecte VID. Mais VID montre vite ses limites lorsqu'on sort du simple formulaire de saisie. Dans ce cas, il faut passer au niveau supérieur qui consiste à définir ses propres styles. On peut également être amené à modifier la gestion des événements afin de les gérer de manière spécifique.

Cet article présente l'utilisation des *flags*, ou en français "drapeaux", qui sont un des éléments constitutifs des styles. Vous allez découvrir le rôle de chacun d'eux et, j'espère, pourrez ainsi en faire bon usage dans vos réalisations.

Cet article est basé sur mes recherches personnelles. N'hésitez pas à me signaler tout problème afin que je puisse l'améliorer au besoin.

Remarque

La description qui suit est basée sur Rebol/View 1.2.48. Je laisse au lecteur le soin de tester les différences entre cette version et les précédentes ou suivantes.

Néanmoins, les différences notables avec View 1.2.1 (dernière version officielle à la date de cet article) seront signalées.

Présentation des *flags*

Les *flags* : c'est (presque) tout bête.

Chaque style (et donc chaque *face*) a une propriété **flags** (c'est un **block !**) dans laquelle sont stockés des mots. Ces mots définissent des caractéristiques de l'objet que View va vérifier et qui vont influer sur la façon de gérer le style.

Il y a plusieurs avantages à cette technique :

- **flags** étant un block, on peut ajouter tout un tas de mots sans avoir à modifier le style en lui ajoutant autant de propriétés.
- On peut changer les **flags** d'un *face* pendant le déroulement du programme et ainsi modifier le comportement de ce *face* selon les besoins.

Définitions

Un **style** est un objet View utilisé comme modèle pour créer des *faces* tel que ceux stockés dans **system/view/VID/vid-styles**. On peut le considérer comme une classe au sens POO.

Un **face** est un objet View créé à partir d'un style. On peut considérer que c'est une instance de classe.

Mais en Rebol, instance et classe sont équivalents.

Gestion des *flags*

Les *flags* étant stockés dans un block, on pourrait les gérer directement avec les opérations **insert**, **remove**, etc... Mais cela ne serait pas très pratique.

Donc Rebol/view nous fournit 3 fonctions pour gérer ces drapeaux :

flag-face : *ajoute si besoin un flag dans le block **flags** d'un face.*

flag-face ? : *teste la présence d'un flag dans le block **flags** d'un face.*

deflag-face : *enlève un flag du block **flags** d'un face s'il s'y trouve.*

En utilisant la fonction **help** sur ces trois fonctions, vous obtiendrez l'usage suivant, que je vous ai francisé :

```
>> help flag-face
USAGE:
    FLAG-FACE face 'flag

DESCRIPTION:
     Met un drapeau dans un face VID.
     FLAG-FACE est une fonction.

ARGUMENTS:
     face -- (Type: object)
     flag -- (Type: any)
```

```
>> help flag-face?
USAGE:
    FLAG-FACE? face 'flag

DESCRIPTION:
     Teste un drapeau dans un face VID.
     FLAG-FACE? est une fonction.

ARGUMENTS:
     face -- (Type: object)
     flag -- (Type: any)


>> help deflag-face
USAGE:
    DEFLAG-FACE face 'flag

DESCRIPTION:
     Supprime un drapeau d'un face VID.
     DEFLAG-FACE est une fonction.

ARGUMENTS:
     face -- (Type: object)
     flag -- (Type: any)
```

On peut noter que le drapeau n'est pas forcément un **word !** puisque le type attendu est *any*.

Remarque

L'argument **flag** est précédé d'une apostrophe, ce qui signifie que cet argument ne sera pas réduit avant d'être transmis à la fonction. Voir le Core Manual pour plus de détails.

Soyons fou et regardons le code source de chaque fonction :

```
>> source flag-face
flag-face: func [
    "Sets a flag in a VID face."
    face [object!]
    'flag
][
    if none? face/flags [face/flags: copy [flags]]
    if not find face/flags 'flags [face/flags: copy face/flags insert face/flags 'flags]
    append face/flags flag
]

>> source flag-face?
flag-face?: func [
    "Checks a flag in a VID face."
    face [object!]
```

```
    'flag
][
    all [face/flags find face/flags flag]
]

>> source deflag-face
deflag-face: func [
    "Clears a flag in a VID face."
    face [object!]
    'flag
][
    if none? face/flags [exit]
    if not find face/flags 'flags [face/flags: copy face/flags insert face/flags 'flags]
    remove any [find face/flags flag []]
]
```

Ici, on notera que les fonctions **flag-face** et **deflag-face** prennent la précaution de dupliquer le block **flags** si celui-ci ne contient pas le mot **flags**, avant de le modifier.

J'avoue ne pas être sûr de la raison qui motive cette précaution. On peut penser que c'est pour éviter de modifier une propriété qui serait partagée par plusieurs *faces*. Mais les seules valeurs qui restent communes à des *faces* lors de leur clonage (avec **make**) sont les **object !** (comme **font**, **feel** ou **edge**). Donc il semble que ce ne soit pas ça, ou alors c'est un héritage d'un comportement différent de **make** à l'aube de la première version de View.

Drapeaux par défaut des styles

Avant de regarder chaque *flag* en détails, faisons une petite analyse des drapeaux par défaut des styles.

La petite fonction suivante va faire ce travail :

```
styles: svv/vid-styles
forskip styles 2 [print [styles/1 mold styles/2/flags]]
```

Et voici son résultat :

```
face []
blank-face []
IMAGE []
BACKDROP [fixed drop]
BACKTILE [fixed drop]
BOX []
BAR []
SENSOR []
KEY []
BASE-TEXT [text]
VTEXT [text]
```

```
TEXT [flags text font]
BODY [flags text]
TXT [flags text]
BANNER [flags text font]
VH1 [flags text font]
VH2 [flags text font]
VH3 [flags text]
VH4 [flags text font]
LABEL [flags text font]
VLAB [flags text font]
LBL [flags text font]
LAB [flags text font]
TITLE [flags text font]
H1 [flags text font]
H2 [flags text font]
H3 [flags text font]
H4 [flags text font]
H5 [flags text]
TT [flags text font]
CODE [flags text font]
BUTTON []
CHECK [check input]
CHECK-MARK [check input]
RADIO [radio input]
CHECK-LINE [flags check input font]
RADIO-LINE [flags check input]
LED [input]
ARROW []
TOGGLE [toggle input]
ROTARY [input]
CHOICE [input]
DROP-DOWN [flags text font]
ICON []
FIELD [field return tabbed on-unfocus input]
INFO [field]
AREA [tabbed on-unfocus input]
SLIDER [input]
SCROLLER [input]
PROGRESS [input]
PANEL [panel]
LIST []
TEXT-LIST [flags text as-is input]
ANIM []
BTN []
BTN-ENTER []
BTN-CANCEL []
BTN-HELP [flags font]
LOGO-BAR []
TOG [toggle input][/CODE]
```

Description de chaque *flag*

Il n'y a malheureusement pas (à ma connaissance) de document officiel sur les différents mots possibles et leur influence. Donc ce qui suit est issu de mes expériences et recherches.

D'abord quelques drapeaux simples :

flags : *est ajouté par les fonctions **flag-face** et **deflag-face** et indique que le block **flags** a été dupliqué et qu'il ne faut pas le refaire (voir **source flag-face**).*

font : *est ajouté par la fonction **set-font** qui sert à modifier la police d'un style. Il indique que l'objet **font** a été dupliqué et que c'est inutile de le faire à nouveau (voir **source set-font**).*

On continue avec 2 *flags* liés à VID :

as-is : *est ajouté par le mot-clé **as-is** de VID et indique que le texte du style ne doit pas être modifié par **trim** lors de son initialisation (voir le block **init** du style **base-text** par exemple).*

hide : *est ajouté par le mot-clé **hide** de VID. Les fonctions de gestion du texte tiennent compte de ce flag afin de masquer les caractères saisis. On voit des "*" à la place. Typiquement utilisé pour la saisie de mot de passe dans un face de type field.*

Ces drapeaux-ci sont en rapport avec les champs de saisie (**field** et **area**) :

tabbed : *indique que le style est accessible par la touche Tabulation (prise/perte de focus).*

return : *indique que l'appui sur la touche Entrée valide le champ et active le champ suivant (prise/perte de focus). Cela revient à dire que le champ n'est pas multi-lignes.*

on-unfocus : *indique que l'action associée au style doit être exécutée lorsque le style perd le focus.*

field : *indique que le texte du champ est sélectionné lorsqu'il reçoit le focus.*

Les *flags* précédents, ainsi que le flag **hide**, sont testés et utilisés par les fonctions de gestion du texte et du focus. Voir l'objet **ctx-text** et particulièrement la fonction **ctx-text/edit-text**. Voir aussi **focus**, **unfocus** et la gestion d'événements de View comme **system/view/screen-face/feel/event-funcs/1**).

Ceux-là sont utilisés par la fonction **layout** (voir **source layout**) :

fixed : *indique que la taille du champ ne doit pas être prise en compte dans le calcul de la position du champ qui suit (on bouge pas quoi). Utilisé pour les fonds **backdrop** et **backtile**.*

drop : *indique que le champ est un fond et que sa taille doit être ajustée à la taille de la fenêtre. Utilisé pour les fonds **backdrop** et **backtile**.*

Celui-ci est un peu du même genre :

text : *est utilisé lors du processus d'interprétation des instructions VID. La fonction **multi/color** du style est appelée pour dispatcher les **tuple !** trouvés dans ces instructions. Si le flag est présent, le premier **tuple !** est considéré comme la couleur de la police. Dans le cas contraire, ce sera la couleur de fond.*

Les trois drapeaux suivants sont utilisés dans le **feel/engage** des styles où on les trouve. Lorsqu'on groupe les styles avec le mot-clé VID **of** (ex : *radio of 'choix1 radio of 'choix1 radio of 'choix2*), le mot qui suit **of** est mémorisé dans la propriété **related** du *face*. Un clic sur ce *face* provoque la recherche dans toute la fenêtre, des *faces* du même type et dont la propriété **related** a la même valeur. Ceci, afin de remettre leur propriété **data** à **false**, celle de l'élément cliqué étant mise à **true**. Ainsi, il n'y a toujours qu'un seul élément d'un groupe qui est "vrai".

check : *indique que le style est un check box.*

radio : *indique que le style est un radio bouton.*

toggle : *indique que le style est un toggle.*

Le suivant est sympa :

input : *a été rajouté dans la béta de View 1.2.34. Il indique les styles qui contiennent une valeur. Les fonctions **get-face** et **set-face** en conjonction avec les accesseurs utilisent ce flag pour mémoriser ou affecter le contenu de tous les champs d'un layout en une seule commande. Vivement que ce soit officiel !!!*

Un exemple d'utilisation des *flags*

Afin d'illustrer le propos, essayons d'utiliser un *flag* personnalisé.

Nous allons mettre en place un système très simple d'activation/désactivation des éléments d'une fenêtre.

Pour cela, nous allons utiliser un *flag* pour marquer les *faces* désactivées. Appelons le **disable**.

Ce drapeau sera vérifié par une fonction dans la chaine de gestion des événements. Si l'élément auquel est destiné l'événement est marqué par le *flag*, l'événement sera ignoré.

Pour commencer voici 2 fonctions permettant d'activer ou désactiver un *face* :

```
active: func [face] [deflag-face face disable]
```

```
desactive: func [face] [flag-face face disable]
```

Maintenant, réfléchissons à la manière de filtrer les événements.

D'abord, à quel endroit le faire ? Deux possibilités : 1) dans le **feel/detect** de la fenêtre 2) globalement, avec **insert-event-func**. La deuxième méthode permet de gérer toutes les fenêtres a un seul endroit, c'est celle que nous allons utiliser.

Tout de suite, la fonction, je vais la détailler ensuite :

```
insert-event-func ev-filter: func [face event] [
```

```
    foreach f event/face/pane [
        if all [within? event/offset f/offset f/size  flag-face? f disable] [
            print [f/var "est désactivé"]
            event: none break
        ]
    ]
    event
]
```

La première ligne crée la fonction **ev-filter** et l'ajoute à la liste des fonctions de gestion d'événements. Comme ne l'indique pas la documentation de View, où elle est juste mentionnée (mais **source** est bien utile là encore), les fonctions à passer à **insert-event-func** reçoivent 2 valeurs : un *face* qui sera toujours **system/view/screen-face** et l'événement (**event**).

A l'intérieur, une boucle **foreach** parcourt les éléments de la fenêtre où s'est produit l'événement. Cette fenêtre, on ne la trouve pas dans **face**, qui représente l'écran, mais dans la propriété de l'événement **event/face**. Les éléments d'une fenêtre sont dans sa propriété **pane**, donc on parcourt **event/face/pane**

Dans cette boucle, on vérifie deux conditions (avec **all**) : est-ce que la souris est sur l'élément en cours grâce à **within ?** et est-ce que cet élément a le drapeau **disable** ? Si la réponse est positive, on met la variable **event** à **none** et on sort de la boucle avec **break**. Au passage on affiche un petit message pour indiquer que l'événement a été ignoré (c'est pour la mise au point).

En sortie, la fonction retourne **event**, l'événement ou **none** s'il est ignoré.

Mettons tout ça avec un petit exemple pour pouvoir tester :

```
Rebol [
    title: "Simple activation/désactivation de face"
    author: "Didier Cadieu"
]

view/new layout [
    across
    text "b1:" b1: button "Bouton" [print face/text] return
    text "b2:" b2: toggle "Basculera" "Basculé" [print face/text] return
    text "b3:" b3: field "Champ" return
    text "b4:" b4: check return
    text "b5:" b5: radio of 'un return
    text "b6:" b6: radio of 'un return
    box 200x2 black return
    button "Active tout" [active-tout] button "Désactive tout" [desactive-tout] return
    c: rotary "b1" "b2" "b3" "b4" "b5" "b6"
    button "Activer" [active get load c/text]
    button "Désactiver" [desactive get load c/text] return
]

active: func [f] [deflag-face f disable]
```

```
desactive: func [f] [flag-face f disable]

;*** La liste de tous les mots représentant un champ de la fenêtre que l'on souhaite
;*** activer/désactiver globalement
b-list: [b1 b2 b3 b4 b5 b6]

;*** Active tous les champs de la liste
active-tout: does [
    foreach b b-list [active get :b]
]

;*** Désactive tous les champs de la liste
desactive-tout: does [
    foreach b b-list [desactive get :b]
]

insert-event-func func [face event] [
    if event/type = 'key [print [event/1 event/2 event/3 event/5 event/6 dump-face event/7]]
    foreach f event/face/pane [
        if all [within? event/offset f/offset f/size flag-face? f disable] [
            print [f/var "est désactivé"] event: none break
        ]
    ]
    event
]

do-events
```

Attention, je ne prétends pas que cette fonction soit parfaite et j'ai prévenu que l'exemple était simpliste. En effet, elle ne tient pas compte des faces imbriquées, comme les *panels*. Elle ne filtre que les événements de la souris. Et il n'y a pas d'indication visuelle de l'état désactivé des éléments.

Pour en savoir plus

Si vous voulez en savoir plus, je vous invite à plonger dans le code de View. Mais pour plonger profond, il faut du matériel, alors penser à utiliser l'excellent **Anamonitor** de Romano Paolo Tenca.

Vous pourrez ainsi aller voir par vous-mêmes comment sont utilisés les différents *flags* détaillés précédemment.

En résumé

Les flags sont donc utilisés à tous les niveaux dans View :

- Par **layout** (***drop*** et ***fixed***) ou des fonctions liées (***text***)
- Par les block **init** des styles de texte (***as-is***)

- Par le **feel/engage** de certains styles "groupables" (***check***, ***radio*** et ***toggle***)
- Par la gestion de l'entrée du texte (***hide***, ***return*** et ***tabbed***) en conjonction avec les fonctions de gestion du focus (***on-unfocus*** et ***field***)
- Comme indicateur d'une action effectué à ne pas refaire (***flags*** et ***font***)
- Pour simplifier le travail du programmeur (***input***)