Requesters, Dialogs and Tool Windows

Table of Contents

1. Dialog System

A dialog design method has been developed for the VID Extension Kit. This is a method which I think is usable for the R3 GUI in that is a generic system for facilitating layouts following specific rules as parts of dialogs and uses fixed outcomes, readable by the user through the window face data.

As a result, the creation of new requesters becomes uniform and trivial and more related to style design than being a special separate discipline and also simplifies creating a large variety of requesters. The use of requesters also is uniform and trivial and the final dialog system in the VID Extension Kit is about 300 lines of code, whereas over 100 of those are reserved for the native file requester, which is not very well implemented.

1.1. Division

The dialog system is divided in several pieces:

- The **layouts**, which is a collection of styles that contain standard layouts for the dialogs. This can scale to any amount of layouts. This is meant for use and extension by dialog developers.

- The <b>dialog buttons</b>, which is a collection of button styles and standardized groups of button styles for use in a variety of dialogs. This is meant for use and extension by dialog developers.

- The <b>dialog functions</b>, which is a small set of functions to build, display, close and collect and handle returned data from the dialogs. This is meant for use and *not* for extension by dialog developers.

- The **localization** strings, which is a block of strings that are used as language resources for the layouts.

- The <b>dialog user functions</b>, which the user uses, when he/she wants to display a dialog. This is only for end users.

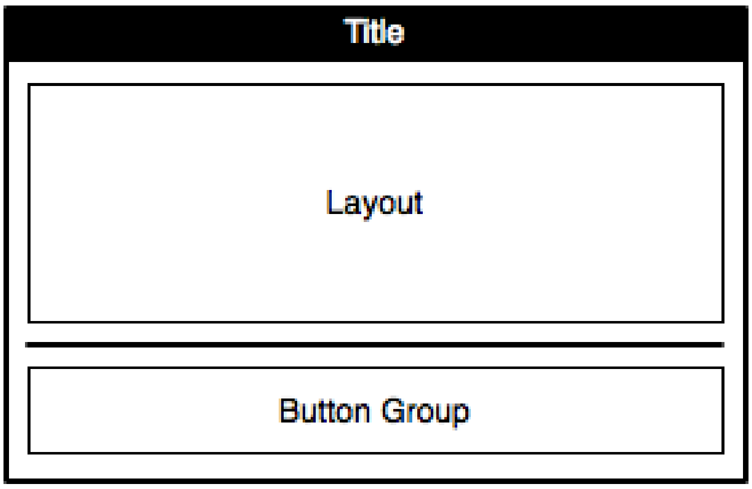
2. Dialog Design

A dialog consists of a single face determined by a word, which represents the name of the style used, and a group of zero or more buttons. A function, `create-dialog` is used to build a dialog and it's then used to build a range of different dialogs.

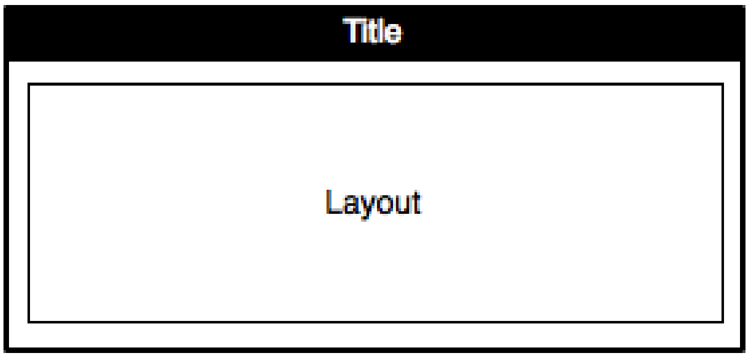
The function accepts the following input:

title	Title of the dialog
layout	The layout to use for the dialog
buttons	A word referring to a specific standard button group, a block to display a custom row of buttons, `none` for no buttons

The outcome is a dialog of this form:



And when no buttons are used:



2.1. Layout Rules

Layouts for requesters are called super styles in the VID Extension Kit. Whether that's necessary here, I don't know, but they follow the concept of simply being styles with larger areas of content. By using styles rather than just layout blocks, we can also build in actors and the correct amount of intelligence for initializing complex layouts and setting and getting data in them.

Some rules to follow:

1. Their content must be entirely settable with one SET-FACE call

2. Their content must be entirely gettable with one GET-FACE call

2.2. Available Layouts

The content can be anything from a single face to a full sized form, and there are certain standard layouts available:

LAYOUT-EMAIL	An email form with validation
LAYOUT-COLOR	Displays a color palette
LAYOUT-VALUE	Displays a single field for requesting a value
LAYOUT-USER	Displays two fields for user/pass combination
LAYOUT-PASS	Displays a single password field
LAYOUT-RENAME	Displays a read-only original name and a field to type in a new name
LAYOUT-LIST	Displays a table of values for selection
LAYOUT-DATE	Displays a month calender for selecting a date
LAYOUT-DIR	Displays a directory tree for selecting a directory
LAYOUT-DOWNLOAD	Displays a progress bar for a single download
LAYOUT-DOWNLOADS	Displays two progress bars for multiple downloads
LAYOUT-FIND	Displays a Find dialog to find text in a text face using ON-SEARCH actor
LAYOUT-REPLACE	Displays a Search and Replace dialog to search and replace text in a text face using ON-SEARCH actor
LAYOUT-QUESTION	Displays a text and a question icon
LAYOUT-NOTIFY	Displays a text and a notification icon
LAYOUT-WARNING	Displays a text and a warning icon (stronger than alert)
LAYOUT-ALERT	Displays a text and an alert icon (stronger than notify)

LAYOUT-ABOUT	Displays a text extracted from the current script header
LAYOUT-FLASH	Displays a text and a small alert icon for flash dialogs

The user is entirely free to extend this list.

2.3. Button Row

The button row would consists of one or more standard buttons. The standard buttons use functions for providing a correct exit route out of the dialog, without having the user to manually perform maintenance procedures when closing the dialog, such as form validation.

The layout would follow that of the OS, as different operating systems use different layouts for the content, TRUE or FALSE exit route from the dialog.

There is going to be two standard buttons:

VALUE-BUTTON	The VALUE button closes the dialog unconditionally and stores the value (`TRUE`, `FALSE`, `NONE` or `lit-word!`) given as an argument to the button.
VALIDATE-BUTTON	The VALIDATE button, based on `TRUE-BUTTON`, first tries to validate the layout and won’t close the dialog, if validation fails. If it validates, the layout data is collected using GET-FACE and stored. The `VALIDATE-BUTTON` contains an indicator, whether the layout content has changed (is dirty) and is eligible for being stored.

See also <b>Dialog Exit Routes</b> for use of these buttons.

Groupings

Various standard button groupings will exist. This is both to allow using a group by passing a single word to the dialog creation function and also to allow setting the order of buttons globally for the operating system.

Furthermore, the groupings will not contain texts directly for the layouts, as we may want to localize them, similarly for layouts. I’m going to describe them using English names below anyway.

Also the grouping can be horizontally or vertically laid out, depending on operating system needs.

The groupings are:

```
close [value-button "Close" none]
ok [value-button "OK" true]
ok-cancel [value-button "OK" true value-button "Cancel" false]
use-cancel [validate-button "Use" value-button "Cancel" none]
save-cancel [validate-button "Save" value-button "Cancel" none]
save-apply-cancel [validate-button "Save" validate-button "Apply" value-button "Cancel"
yes-no [value-button "Yes" true value-button "No" false]
yes-no-cancel [value-button "Yes" value-button "No" false value-button "Cancel" none]
retry-cancel [value-button "Retry" true value-button "Cancel" none]
```

Custom Groupings

The create-dialog function will allow passing a block as a custom grouping.

Example:

```
[
	value-button "Demo" 'demo
	value-button "Enter License" 'license
	value-button "Quit" 'quit
]
```

2.4. Dialog Exit Routes

While you can use one of 4 values for a `VALUE-BUTTON`, their use determines the outcome of the dialog.

All dialogs share common routes out:

1. <b>Exit with `none` value</b>, which means to not bring back information from the dialog. The user can test the outcome simply by checking for NONE from the window face or passed word. This is the only outcome and all dialogs support this outcome. This is equivalent to:

   - Pressing a `VALUE-BUTTON` with `none` value attached or similar button in the dialog
   - Pressing the `ESCAPE` key
   - Letting the dialog time out

2. <b>Exit with a `false` value</b>, which means to not bring back information from the dialog. The user can test the outcome simply by checking for FALSE from the window face or passed word. This is equivalent to pressing a `NO` button in the dialog.

3. <b>Exit with a `true` value</b>, which means not to bring back information from the dialog, other than pressing the equivalent to `YES` or similar button in the dialog. Various simple question dialogs support this.

4. <b>Exit with a WORD value</b>, which means to bring back a single value from the dialog, by pressing a WORD-BUTTON in the dialog. Custom multiple-answer dialogs will support this.

5. <b>Exit with content value</b>, which means to bring back information from the layout of the dialog. This is done by pressing the `SAVE`, `USE` or similar button in the dialog. As a bonus, this route will perform layout validation, if any validation reactors are present. If the layout is found to have invalid content, it will be marked up and the dialog is not closed.

2.5. Creating Dialogs

When showing a dialog, it needs to be created from scratch using `create-dialog`. By passing the **layout**, the **content** and the **buttons** grouping to `create-dialog`, the dialog is formed and the window is displayed.

Code example:

```
request-color: func [title value buttons] [create-dialog 'layout-color value 'use-cancel
```

2.6. Using Dialogs

The way to use a dialog is by one of the functions created by the create-dialog function. The usage of a dialog involves choosing the dialog to display and where to store the result if needed.

```
request-color/in 240.0.240 my-color
```

3. Tool Window Design

Tool windows are designed identically to regular requesters and dialogs except for a missing row of buttons at the bottom. Tool windows return values through a call-back in realtime to whatever is requesting the value. An example of this is a color droplet, which has opened a color palette tool window.

Under various operating systems, it should be possible to display tool windows using the native tool window frame.