Making New Themes For Kvantum

Please install Kvantum, read the file “Theme-Config”, and create the configuration folder before reading this document! A basic knowledge of Inkscape is also presupposed here.

Making new themes may take time but its logic is not complex. Each Kvantum theme consists of a configuration file (explained in “Theme-Config”) and an SVG image. Both should have the same name – ***MY_THEME***, for example – and be put into the same folder *“~/.config/Kvantum/**MY_THEME**”*. This document explains how you could create an SVG image for your new theme.

But let us first try an alternative theme that is already included in the source. Its name is ***Glassy***. If you look into the folder *“doc/**Glassy**”* you will find the config file *“**Glassy**.kvconfig”* and the SVG image “***Glassy**.svg”*. Just put the folder “Glassy” into *“~/.config/Kvantum/”* and create the file “kvantum.kvconfig” in the latter directory with this line in it:

theme=***Glassy***

Now run any Qt or KDE application and see the difference:

Scrollbar and buttons have a glassy look with rounded edges, tabs, progressbars and line-edits also have rounded edges, tabs are left aligned and the active tab is attached to the tab widget. If you open “*Glassy.svg”* with Inkscape, you will find just a few objects in it. Kvantum first searches that image for the widget parts and if it does not find the relevant object names, it will go to the image of the default

theme. That is similar to what Kvantum does with the configuration files, as was explained in "Theme-Config".

For instance, to not show scrollbar grip indicators of the default theme, invisible rectangles with names "grip-normal", "grip-focused" and "grip-pressed" are created in *"Glassy.svg"*. On the other hand, in the same image, there is no object for the interior of progressbar patterns but just objects for their frame, so that the new frame is used alongside the default interior.

The number of objects you create inside your SVG image depends on how much you want your theme to be different from the default one. The easiest way is to start with the default SVG image itself. The file *"default.svg"* in the "doc" folder is the image for the default theme. It contains useful comments on various objects. *(Please do not use the image with the same name in the folder "style/themeconfig/" because it is cleaned with **SVG Cleaner** and not only does not contain any comment, the groupings of its objects could also be misleading!)* You could change the objects one by one in whatever way you prefer, delete those objects you do not want to change, put invisible rectangles in place of those you want to omit, and even add new objects.

Do not forget that the look of your theme is determined by its config file too. Also note that your theme will be used together with a color scheme of your choice. Therefore, select colors and gradients carefully, so that they match your color scheme. Yes! There may be a lot of work to do but it is what you pay for being able to control virtually every aspect of each widget.

After you have finished your work with the image, first back it up and then, preferably, clean it with *SVG Cleaner*. It is a nice tool that can reduce the size of an SVG image considerably. In this way, the memory footprint will be minimized. If *SVG Cleaner* is not in the repository of your Linux distro, you could get its latest source from *https://github.com/RazrFalcon/SVGCleaner*.

To make your theme available to others, put these three files in a folder named ***MY_THEME***:

MY_THEME.svg (the SVG image)
MY_THEME.kvconfig (the Kvantum config file)
MY_THEME.colors (a KDE color scheme)

In this way, your users could install and choose your theme easily with "Kvantum Manager", which is a simple GUI made for that purpose.

That is the basic logic behind making themes for Kvantum. Now, we pay attention to some details.

Elements

Each section of the config file – except for the General and Hacks sections – determines the look of a widget by setting the elements that are used to draw it (see Sections Table in "Theme-Config"). Usually, there are three kinds of elements, namely, *frame*, *interior* and *indicator*. Some widgets may not need all of them and some may need more.

The basic names of elements are mostly optional but there are a few exceptions, i. e. the names of dial elements (dial, dial-notches, dial-handle) MDI titlebar buttons (mdi-maximize, mdi-restore, mdi-

minimize, mdi-close, mdi-shade, mdi-menu), and the default button indicator (button-default-indicator).

Interior and Frames

The *names* (or *id* strings) of the rectangular objects, that are used to draw the frame and interior elements of a widget, depend on its state. There are five states at most: *normal*, *focused*, *pressed*, *toggled*, and *disabled*. For each state, there are at most ten rectangular objects: one for the interior and eight for the frame.

Each interior object should have a name (id) with this format:

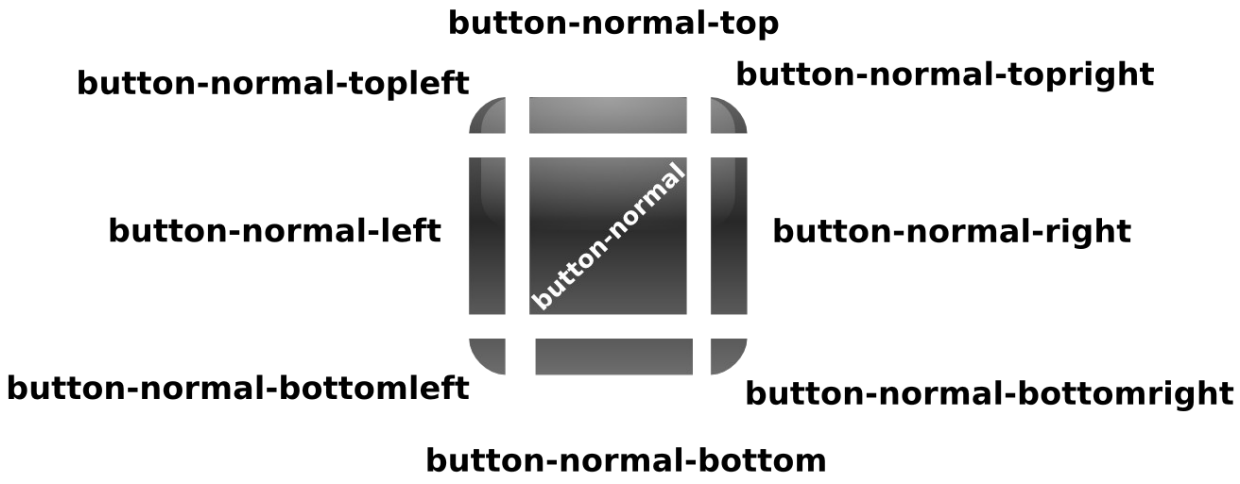
iNAME-STATE

Here, iNAME is set as the string value of "interior.element" in the config file, and STATE is the state of the widget the object represents. The name of each frame object should be:

fNAME-STATE-POSITION

Where fNAME is set as the string value of "frame.element" in the config file, and POSITION could be *top*, *bottom*, *left*, *right*, *topleft* , *topright* , *bottomleft* or *bottomright*.

For example, the following image shows the names of the nine objects that together draw the normal state of a button widget, whose frame and interior names are both "button". It could be defined in the "PanelButtonCommand" or "PanelButtonTool" section of the config file.



Indicators

An indicator is a sign or icon on a widget that shows some action is available or informs the user of something about that widget.

For instance, some tool buttons have “arrow indicators,” which will show a drop-down menu if pressed. Arrow indicators also appear on combo boxes to show that other options are available. Or the handle of a scrollbar slider may have an indicator that can make it easier to find it. The close button of a tab can be seen as an indicator too. And so on.

This is the list of all indicators Kvantum draws:

Section	Indicator
TreeExpander	Indicators showing that a tree branch is expanded or not.
IndicatorSpinBox	Up/down and plus/minus indicators for spin widgets.
HeaderSection	The sorting indicator for headers in item views.
DropDownButton	Indicator that shows a drop menu is available.
Tab	Tab close button and also tab-tear indicator.
IndicatorArrow	Left/right/up/down arrows used in various widgets.
Scrollbar	Indicators for scrolling.
ScrollbarSlider	A decorative indicator on the slider of a scrollbar. Also glows at its top and bottom.
Toolbar	The handle of a floatable toolbar; also toolbar separators.
SizeGrip	Window resize indicator.
PanelButtonCommand	An indicator for the default push button, another one showing that the button has a drop menu, and also arrow indicators for tool buttons.
SliderCursor	That handle of a slider (a volume control, for example).
TitleBar	Maximize/restore/minimize/close/shade/menu buttons of the titlebar of a QmdiSubWindow.
MenuItem	The tear-off indicator for detachable menus; also the menu-item separators and submenu/scroller arrows. As an exception, if there is no submenu/scroller arrow element for menu-items in the SVG image, those of IndicatorArrow will be used.
Splitter	An indicator for the handle of a splitter. Also a decorative indicator on it.

States

As mentioned before, there are five states at most: *normal*, *focused*, *pressed*, *toggled*, and *disabled*. You do not need to draw any object for the disabled state of interiors or frames because they are automatically created based on the normal state by reducing its opacity.

However, the disabled states of all *menu-items* and *menubar-items* and also those of most *indicators* should be included in the SVG image because, for example, we may want disabled indicators to be totally invisible or have a neutral color.

Not all widgets have all the possible states. For example, toolbars only have the normal and disabled states, line-edits can only be in a normal, focused or disabled state, etc. You could know about the possible states by examining the image *"doc/default.svg"* with Inkscape. Not drawing redundant objects not only saves your time but also reduces the memory usage of your theme.

Orientations

Some widgets, like scrollbars, can be oriented both vertically and horizontally; some others, like tabs, have even more orientations. Even if you use gradients, you will need to draw objects only for one of the possible orientations, which may be different based on which orientation a widget most commonly has in various applications. There is no consensus about that but these are the orientations you should use when you draw objects for Kvantum:

Widget	Orientation
Scrollbar (slider, groove, indicator, grip)	Vertical
Slider groove (like in volume controls)	Vertical
Slider Handle	Vertical with tick marks to the right of the slider. *(The handle will be rotated or mirrored only if its width and height are different.)*
Splitter Handle	Vertical (which means that the splitter itself is horizontal technically)
Progressbar (groove, pattern/indicator)	Horizontal
Tab	Horizontal (and top)
Toolbar	Horizontal
Toolbar Handle (for floatable toolbars)	Vertical (the toolbar itself is horizontal)
Toolbar Separator (between toolbar buttons)	Vertical (the toolbar itself is horizontal)

Kvantum automatically draws the other orientation(s) for each of the above widgets. There is only one exception and it is the arrow indicators of the "IndicatorArrow" section, all of whose orientations should be included. It is better to draw all orientations of the arrow indicators of the "MenuItem"section too but if they are missing, those of "IndicatorArrow" will be used.

Inactiveness

The window containing a widget may not have keyboard focus, in which case it is said to be inactive. Inactive windows are usually distinguished from the active one by their title-bars.

In Kvantum, inactiveness of a widget means that its window is inactive. Inactiveness can be considered as a sub-state so that, for each state of an SVG object, an "inactive" counterpart can be added. The name of such objects should have the string *"-inactive"* after their state strings. For example:

E-normal**-inactive**(-top/-bottom/...)
E-focused**-inactive**(-top/-bottom/...)
E-pressed**-inactive**(-top/-bottom/...)
E-toggled**-inactive**(-top/-bottom/...)
E-disabled**-inactive**(-top/-bottom/...)

Where "E" is the name of the element that object draws, as it appears in the config file.

This feature is completely optional and is not used in the default theme. If "inactive" objects are present, they will be used for drawing widgets on inactive windows; otherwise the usual objects will be used for drawing widgets on both active and inactive windows.

Button Icons

The icons for some standard buttons can be added as SVG elements. Their names are as follows:

dialog-cancel
dialog-ok
dialog-ok-apply
folder-open
folder-new
document-save
go-previous
go-next
go-up
go-down

When one of these elements is missing, it will be taken from the currently used icon theme if it exists and the application is aware of Desktop settings; otherwise, the corresponding Qt icon will be used.

***** ***** *****

Anyway, by renaming *"default.svg"* to *"**MY_THEME**.svg"*, putting it alongside the file *"**MY_THEME** .kvconfig"* in *"~/.config/Kvantum/**MY_THEME**"*, and playing with them, you could learn more about theme-making than by reading any document.

After every change you make to the SVG image or config file, you could see how various widgets look by clicking on the **Preview** button of **Kvantum Manager** or by entering the command *kvantumpreview* in terminal.