Making New Themes
For Kvantum

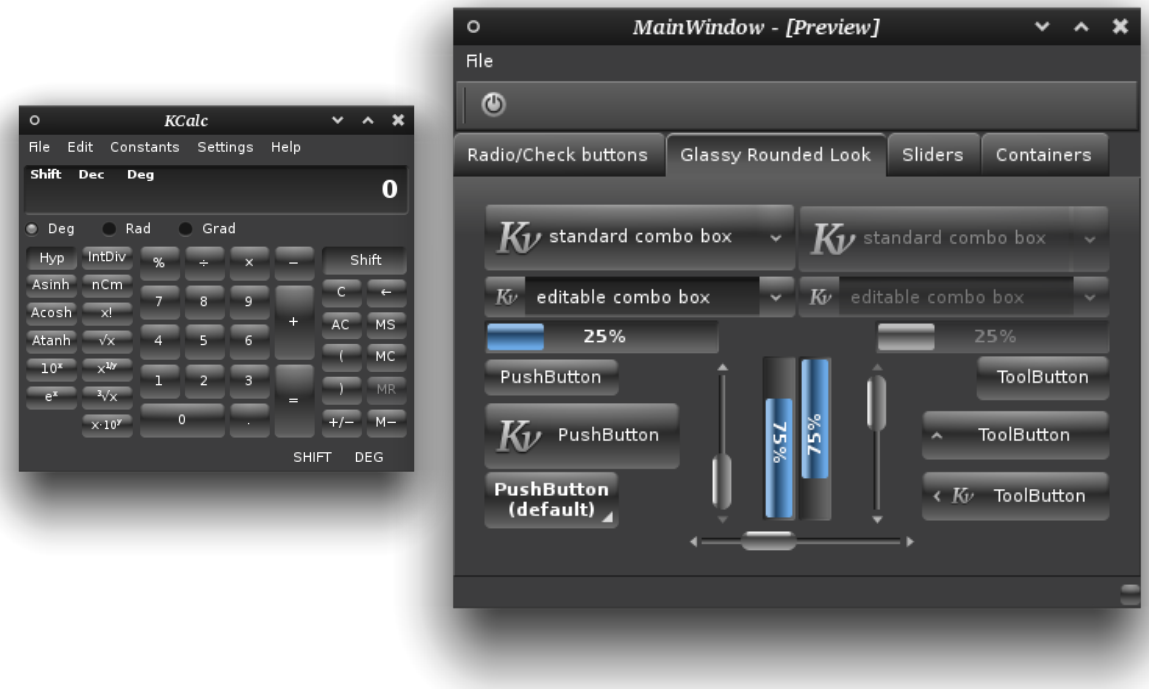
Please install Kvantum, read the file “Theme-Config”, and create the configuration folder before reading this document! A basic knowledge of Inkscape is also presupposed here.

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***. The folder *“doc/**Glassy**”* contains the configuration file *“**Glassy**.kvconfig”* and the SVG image “***Glassy**.svg”*. If you install ***Glassy*** and change the active theme to it with **Kvantum Manager**, the folder “Glassy” will be created inside *“~/.config/Kvantum/”*, the above-mentioned files will be put into it, and the file “*~/.config/Kvantum/kvantum.kvconfig*” will contain:

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". However, because of the concept of "state", there is a difference:

SVG Object Inheritance Rule:

When an SVG object is missing from the SVG file of a theme,
(1) If it has no state, the object with the same name from the default theme will be used; but
(2) If it has a *focused, pressed or toggled* state, the *normal* state object from the same SVG file will be used and only if it does not exist either, the object with the same name from the default theme will be used.

Back to *Glassy*, for not showing 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 by **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 configuration 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 finished your work with the image, first back it up, remove all of its comments, clean it up (Inkscape → File → clean up document) and then, preferably, clean it with SVG Cleaner too. SVG Cleaner 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. If you have used the cloning menu-item of Inkscape (which links similar objects) as far as possible, the image will have the minimum size.

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 configuration 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.

There are also other ways of making theme folders, especially when your Kvantum theme is a part of a more comprehensive theme package with the same name, which, for example, includes GTK+ themes

too. For more information, see Theme Installation Paths!

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

Elements

Each section of the configuration file – except for the General, GeneralColors 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*), the default button indicator (*button-default-indicator*), and the header separator (*header-separator*).

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 nine rectangular objects: one for the interior and eight for the frame.

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

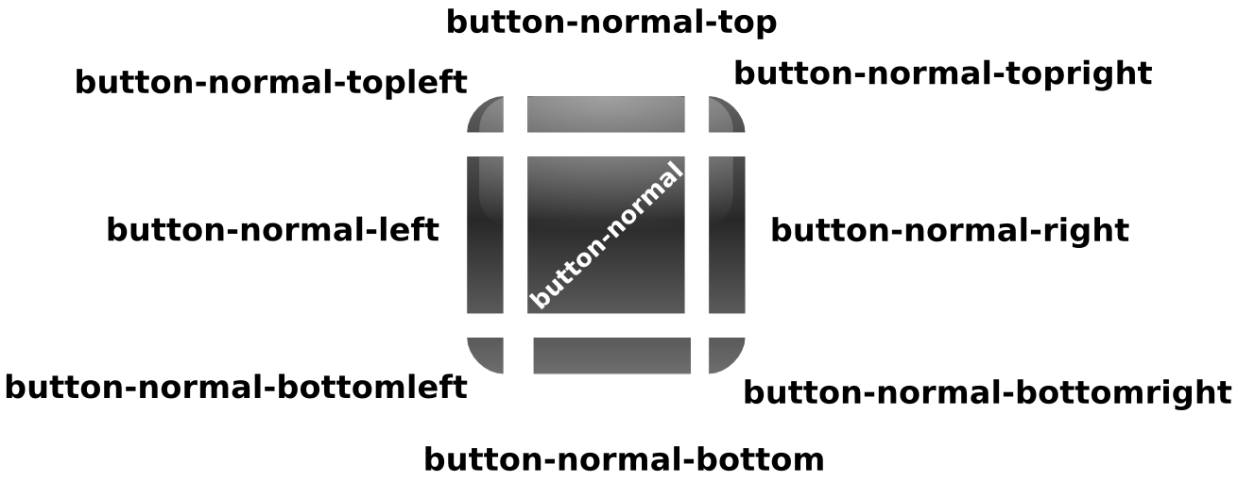
iNAME-STATE

Here, iNAME is set as the string value of "interior.element" in the configuration 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 configuration 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 configuration file.



Not all widgets have frame. For example, windows do not have frame and it will make no difference whether you set *"frame=true"* and define a frame element under the *Window* section.

Also some framed widgets may not exactly obey the frame widths provided under their corresponding sections. For example, the frame of a tooltip has a uniform thickness on all sides, which is equal to the maximum of the frame widths defined under the *Tooltip* section. Or if there is not enough space for a button, all of its frame widths will be set to 3px at most.

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.

The element name of an indicator is (made out of) the string value of "indicator.element" under the relevant section of the configuration file. Its SVG object name is made by adding its state to its element name with a dash, as in case of interior and frames.

The following table is a list of all indicators Kvantum draws and also their possible states. Some of them are simple indicators, some are complex ones consisting of multiple elements whose names are derived from the same string, and yet some others are complete elements with interior, frame and indicator parts. Here, "indicator base name" means the string value of "indicator.element" under the relevant section. For simple indicators, it is the indicator element name itself.

Section	Indicator
TreeExpander	A sign that shows whether a tree branch is expanded or not. It is a complete element with interior, frame and indicator parts. The names of its indicator elements are made by adding *"-minus"* and *"-plus"* to its indicator base name. Its indicator can have normal, pressed, focused and disabled states.
IndicatorSpinBox	Up/down and plus/minus indicators for spin widgets. Their element names are made by adding *"-up"*, *"-down"*, *"-left"*, *"-right"*, *"-plus"* and *"-minus"* to the indicator base name of IndicatorSpinBox. They can have all possible states. Also an optional separator between the line-edit and horizontal buttons. It consists of top, middle and down objects, whose names are made by adding the strings *"-separator-top"*, *"-separator"* and *"-separator-bottom"* to the indicator base name of IndicatorSpinBox, respectively. Their width is the width of the left frame and the heights of the top and bottom objects are the heights of spinbox's top and bottom frames, respectively. They can have normal, focused and pressed states. The top and bottom objects are drawn only if the normal middle object exists.
HeaderSection	The sorting indicators for headers in item views, whose element names are made by adding *"-down"* and *"-up"* to the indicator base name of HeaderSection. They can have normal, pressed, focused and disabled states. Also the header separator, whose SVG object name is always *header-separator*. If no header separator is included in the SVG image, the right frame of the HeaderSection (or the left frame if the layout is right-to-left) will be used as a separator. The separator width is always equal to the right frame width (or the left frame width for right-to-left layouts).

DropDownButton	An indicator that shows a drop menu is available. It can have all possible states (but see IndicatorArrow below).
Tab	Tab close button and also tab-tear indicators. Their indicator names are made by adding *"-close"* and *"-tear"* to the indicator base name of Tab, respectively. The close indicator can have normal, pressed, focused and disabled states but the tear indicator is stateless. If an element with the toggled state is found for the close indicator in the SVG image, it will be used on the active tab. This may be useful when the background of the active tab has a high contrast with that of normal tabs. Also see Joined Tabs and Tab Separators!
IndicatorArrow	Up/down/left/right/ arrows used in various widgets. Their indicator names are made by adding *"-up"*, *"-down"*, *"-left"* and *"-right"* to the indicator base name of IndicatorArrow and they can have all possible states but the *toggled* state can be omitted, in which case the *pressed* state will be used instead (as an exception to the SVG Object Inheritance Rule).
Scrollbar	Add-line and sub-line indicators for scrolling. Their indicator names are made as in the case of IndicatorArrow above.
ScrollbarSlider	A decorative indicator on the slider of a scrollbar. It can have normal, pressed and focused states.
ScrollbarGroove	Glows at the top and bottom of the scrollbar but inside its groove interior. Their indicator names are made by adding *"-topglow"* and *"-bottomglow"* to the name of the interior element of ScrollbarGroove and they can only have the normal state (but can also have "inactive" counterparts). Their extent is always twice the scrollbar thickness.
Toolbar	The handle of a floatable toolbar, whose indicator name is made by adding *"-handle"* to the Toolbar indicator name. It has no state. If the key *"center_toolbar_handle"* is set to true under the General section, its width and height will both be equal to twice the toolbar indicator size but not less than 8 px; otherwise, its width will always be 8 px and its height will be equal to the toolbar interior height (for horizontal toolbars). Also the toolbar separator, whose indicator name is made by adding *"-separator"* to the Toolbar indicator name. It has no state either. Its thickness is the toolbar indicator size but not less than 4 px.
SizeGrip	The window resize indicator with a maximum size of 13px. Its states are only normal and focused and it should be drawn for the right bottom corner.

PanelButtonCommand	An indicator for the default push button, whose SVG object name is always *button-default-indicator*. Also another indicator showing that the button has a drop menu, whose indicator name is made by adding *"-down"* to the indicator name of PanelButtonCommand. It can have normal, pressed, focused and disabled states.
PanelButtonTool	Arrow indicators. Their indicator names are made as in case of IndicatorArrow above. **Note:** By choosing the same name ("arrow", for example) for the indicators of TreeExpander, IndicatorSpinBox, HeaderSection, IndicatorArrow, Scrollbar, PanelButtonCommand and PanelButtonTool, you could use the same set of SVG elements for all of them. However, that is optional.
SliderCursor	That handle of a slider (a volume control, for example). It is a complete element with interior, frame and indicator parts. Its states can be normal, pressed, focused and disabled.
TitleBar	The maximize/restore/minimize/close/shade/menu indicators of the titlebar of a QmdiSubWindow, whose names are made by adding *"- maximize", "- restore", ..., "-menu"* to the indicator base name of TitleBar respectively. They can have the normal, focused, pressed and disabled states, except for the menu indicator, that can only have the normal state.
MenuItem	The tear-off indicator for detachable menus, whose indicator name is made by adding *"-tearoff"* to the indicator base name of MenuItem. It only has normal and focused states and is repeated every 20px horizontally. Its height is always 8px (use object transparency to make a thinner indicator). Also the menu-item separator, whose indicator name is made by adding *"-separator"* to the indicator base name of MenuItem. It has no state and its height is always 10px (use object transparency to make a thinner separator). Also the submenu/scroller arrows, whose indicator names are made as in case of IndicatorArrow and whose states can be normal, pressed, focused and disabled. **Note:** 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. It is a complete element with interior, frame and indicator parts. Its states are normal, focused and pressed. Here, *indicator.size* gives the indicator height for vertical splitters

	(the width being given by *splitter_width* under the *General* section)
ComboBox	The ComboBox (arrow) indicator is used *only if* its normal state exists; otherwise, the DropDownButton indicator will be used by combo boxes too. Here, the indicator name is the same as the indicator base name and it can have all possible states.
LineEdit	Line-edits do not have any indicator *unless* the key *combo_as_lineedit* is set to true (see "Theme-Config"), in which case their indicator will be used by their containing editable combo boxes *only if* its normal state exists; otherwise, the DropDownButton indicator will be used by editable combo boxes too. The indicator name is the same as the indicator base name and it can have normal and focused states.
CheckBox and RadioButton	Contrary to other indicators, they are just interiors whose names are made by adding *"-normal"*, *"-focused"*, *"-checked-normal"* and *"-checked-focused"* to the interior names of CheckBox and RadioButton. The Checkbox has also two extra elements, whose names are made by adding the strings *"-tristate-normal"* and *"-tristate-focused"* to its interior name. They can also have "inactive" counterparts. Also see Menu Check Boxes and Radio Buttons!
Dial indicators (without section)	They are stateless and their names are always "dial", "dial-notches" and "dial-handle".
Focus Frame	It is usually a frame, under the *Focus* section, that is drawn around some widgets with keyboard focus, but can also have a translucent interior. It has no state. Its frame thickness is at most 2px, regardless of the frame widths under the *Focus* section.

Also see "Orientations".

"Flat" Indicators and High Contrast

Like other widgets, tool or push buttons can have different backgrounds and their text colors should be set appropriately to have enough contrast with their background colors. But unlike other widgets, tool and push buttons can be flat, in which case no background (panel) is drawn for their normal state. In such cases, Kvantum automatically sets their text color to that of the widget behind them (toolbar, menubar or any container whose text color can be set). It will also use "flat" indicators instead of the usual ones *if* they exist in the SVG image and *if* the usual indicators do not have enough contrast with the widget behind flat buttons.

The names of flat indicator objects are made by adding the string **"flat-"** to the beginning of the names of usual indicators. For example, if the name of the indicator element under the *PanelButtonCommand* section is "arrow", extra objects with names "***flat-****arrow-up-normal"*, "***flat-****arrow-down-normal"*, …, "***flat-****arrow-right-focused", "****flat-****arrow-left-pressed"*, etc. could be added to the SVG image. Also the

default button indicator, whose name is always *button-default-indicator*, can have a flat counterpart with the name ***flat-button-default-indicator***.

If Kvantum does not found the "down-normal" objects of "flat" indicators, it will use the usual ones for drawing the indicators of flat buttons. "Flat" indicators are good only when there is a high contrast between the background color of buttons and that of widgets behind them, for example, when dark buttons with white texts and indicators are used together with light containers.

To determine whether there is a high contrast, Kvantum relies on the value of *text.normal.color*. Therefore, ***apart from textless widgets, only the interior elements of those widgets that accept state-specific text colors can have a high contrast with the window or base background.*** For example, menubars, toolbars and buttons have state-specific text colors but generic frames or tab frames do not (see the note in the explanation of the key *text.normal.color* in the file *Theme-Config.pdf)*. As a result, the interior colors of toolbars, menubars or buttons can have a high contrast with the window color (the value of *window.color* under the section *GeneralColors*), provided that their normal text colors are set correctly in the configuration file. However, the interior colors of generic frames or tab frames (if they have any interior element at all) should NOT have a high contrast with the window color because *text.normal.color* has no meaning for them.

Flat indicators are only needed by light-and-dark themes (a light theme with a dark highlight color, for example). They are not limited to tool and push buttons and can be used wherever the background color changes in such a way that it has a high contrast with its usual value. Although buttons only need the normal state of their flat indicator, depending on the light-and-dark theme, the SVG file may need to include all states of flat indicators.

Menu Check Boxes and Radio Buttons

Check boxes and radio buttons can be in menus. Their appearance in menus are like in other places, except that they may be drawn a little smaller. But if you like to give a different appearance to check boxes and/or radio buttons in menus, you could simply add SVG objects, whose names are constructed by adding ***"menu-"*** to the beginning of the names of ordinary check box and radio button objects. If a ***"menu-"*** check or radio object is missing, Kvantum will use its corresponding ordinary object.

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 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, menu-items and menubar-items do not have normal and disabled states; toolbars only have the normal and disabled states; and line-edits can only be in a normal, focused or disabled state, etc. On the other hand, the SVG elements used for drawing

frame focus rectangle (under the *Focus* section) cannot have any state because they are used for distinguishing some widgets that already have keyboard focus.

You could know about the possible states by examining the image *"doc/default.svg"* with Inkscape. For possible states of indicators, see their table. Not drawing redundant objects not only saves your time but also reduces the memory usage.

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
Header	Horizontal
Header Separator (between header sections)	Vertical (the header itself is horizontal)
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 (the values of the keys **slider_handle_width** and **slider_handle_length**) 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)
SizeGrip	To be drawn for the right bottom corner.

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 may be added. The name of such objects should have the string *"-inactive"* after their state strings. For example:

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

Where "E" is the name of the element that the object draws, as it appears in the configuration file. However, the *pressed* and *focused* states cannot have *inactive* counterparts because widgets are pressed or have focus only inside active windows.

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.

Please also note that, since there is no inactive text color, the background of an inactive element should not have a high contrast with that of its active counterpart.

The Default (Push) Button

The default push button is the one that has the keyboard focus. There are two (optional) ways to make it distinct: (1) adding a default button indicator; and/or (2) giving a default frame and/or interior to it.

The default button indicator has no state and the name of its SVG object is always ***"button-default-indicator"***. When drawn on the button, its size is equal to the value of the key *"indicator.size"* under the *PanelButtonCommand* section and its place is on the right or left bottom corner of the widget for LTR or RTL layout direction respectively.

The names of the default frame and interior SVG objects are made by adding the string ***"-default"*** to the end of the name of button elements under the *PanelButtonCommand* section. For example, if the latter is "button", the default button SVG objects should be named as *"button-default"*, *"button-default-top"*, *"button-default-topleft"*, *"button-default-topright"*, etc. The interior object (*"button-default"*, in this example) is better to be semi-transparent, and you could omit it if you want to have only a default frame. Like the default indicator, the default frame and interior do not have any state.

Toolbar Buttons

Toolbar buttons are tool buttons that are situated on toolbars. Therefore, they get their appearance from the section *PanelButtonTool*. But sometimes, you might want to distinguish between toolbar buttons

and other tool buttons visually. For example, you may have used a dark toolbar with a light theme and want toolbar buttons to be dark too. You could use the section *ToolbarButton* for that.

The section *ToolbarButton* gets all of its variables from *PanelButtonTool*, except for text colors, text shadow, and elements (i.e. indicator, frame and interior). So, you could add this section to the config file and only set the values of those keys under it. Like other keys, they can get their values by inheritance too. Setting the values of other keys, such as *frame.bottom* or *text.margin.top*, would have no effect under the section *ToolbarButton* because their values are taken from the *PanelButtonTool*.

Inheritance and Alignment

Although the key *inherits* can be used under various widget sections for not repeating identical properties, its use can result in an interesting visual effect too. For example, if this key is used under the *ComboBox* section as "inherits=PanelButtonCommand", and provided that no frame width or text margin is specified, the frame widths and text margins of combo-boxes will be equal to those of push-buttons. As a result, if a combo-box is located adjacent to a push-button horizontally and if both of them either have icon or are iconless, they will look aligned; in other words, their top as well as bottom borders will be on the same level. The same thing can be said about tool-buttons (under the section *PanelButtonTool*), line-edits and spin-boxes.

Therefore, if you want the horizontally adjacent widgets to look aligned as far as possible, you could rely on the key *inherits* and set it to PanelButtonCommand, as the best candidate for inheritance.

Patterns

A pattern is an image used for tiling the interior of an element. The interior is tiled by it when, at least, one of the keys *interior.x.patternsize* or *interior.y.patternsize* has a positive value under the corresponding section. (The absence of these keys means no tiling.) These keys show how the interior is tiled by the pattern in the horizontal and vertical directions, respectively. If one of them is zero, there will be no tiling in its direction.

The SVG object used for tiling is the interior object itself unless there is another SVG object whose name is made by adding ***"-pattern"*** to the end of the name of the interior object, in which case, the pattern is drawn as a tiled layer over the interior background. Needless to say, in the latter case, the pattern object should have some translucency for the background to be seen behind it.

For example, if the Window section has an interior element called "window" and if, at least, one of the pattern sizes is greater than zero, the SVG object "window-normal" will be used for tiling the background of windows and dialogs (see Interior and Frames). However, if an object with the name "window-normal-pattern" is also present, the background will be drawn by "window-normal" *without tiling* and then, "window-normal-pattern" will be used for tiling over it.

Also note that some widgets never accept patterns, even when the pattern size keys have positive values for them. They are widgets, like grouped toolbar buttons, spinbox buttons and view-items , for which a pattern does not have much meaning.

Tinting Colors

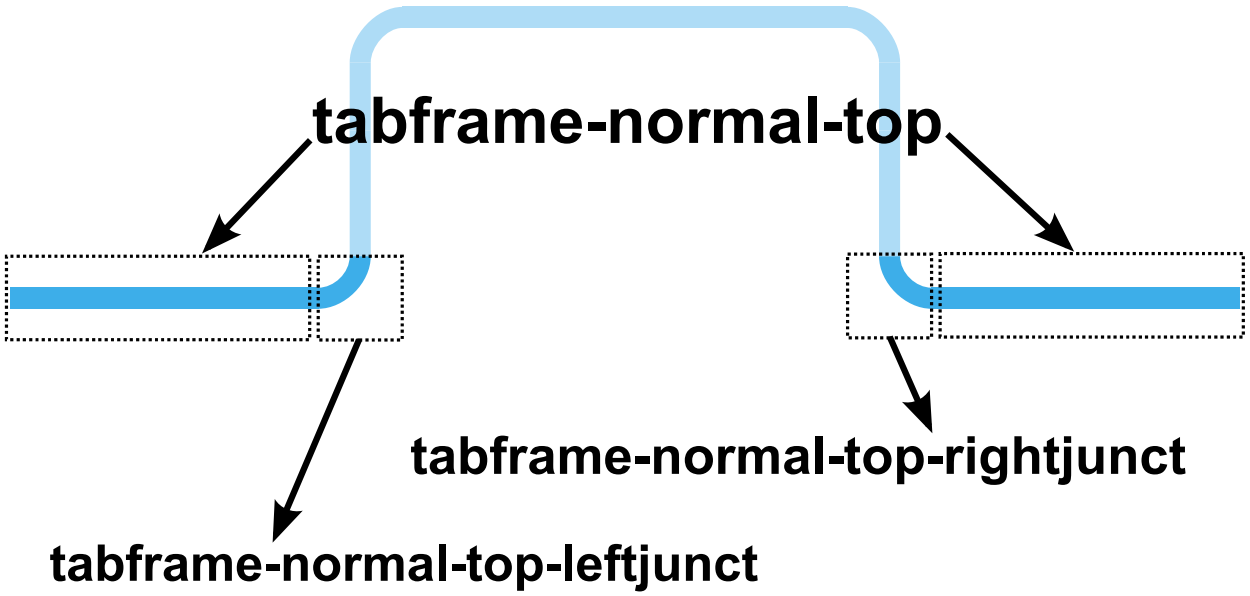
Under the *Hacks* sections, there are two keys for color tinting, namely, *tint_on_mouseover* and *no_selection_tint*. The former tints the label colors on mouseover with the highlight color by the percentage of its value and may be useful with monochrome icons. Usually, it does not need to be set by the theme but can be customized by the user.

The second key determines whether the label icons should be tinted with the highlight color when selected. It may need to be set to *true* with dark-and-light themes. The reason is that there are monochrome icon sets that reverse the color of icons in the "selected" mode and, with dark-and-light themes, Kvantum may change the icon mode to "selected" for appropriate icons to be used on dark or light backgrounds. In such cases, if *no_selection_tint* is not set to *true* explicitly, ordinary icons might be tinted with the highlight color when *not* selected. Therefore, it is always a good idea to set *no_selection_tint* to *true* with dark-and-light themes.

Junctions between Attached Active Tabs and Tab Widgets

If the key *attach_active_tab* is true, active tabs will be attached to their tab widgets. In fact, the frame element from the section *TabFrame* will be cut under an active tab. Of course, the bottom frames of active tabs should be drawn in such a way that they appear really attached to their tab widgets. If the frames in the sections *Tab* and *TabFrame* are thin, everything will be all right. But in the case of thick frames or when you want to customize the two junctions between the left/right frames of the tab and the cut frame of the tab widget, you could add extra SVG objects, whose names are those of the frame elements of *TabFrame* plus the two strings "**-leftjunct**" and "**-rightjunct**".

Provided that the frame name under TabFrame is *tabframe*, the following image shows two of these extra objects for the top frame of a tab widget.



In this image, the dotted rectangles show whole frames, which are drawn in such a way that only their blue part is visible.

Three other pairs of such objects for left, right and bottom frames should also be drawn appropriately if such junctions are used at all.

"Floating" Tabs

In some applications, the tab-bar has no tab widget or the latter is in the "document mode", so that tabs seem "floating". If you have used tab junctions or chosen shapes suitable for attaching tabs to a tab widget, you might want to choose different shapes for the tab interior and (bottom) frames in such cases. You could do so by adding another set of objects, whose names are made by adding the string **"floating-"** to the beginning of the names of the original tab objects.

For example, if the interior and frame elements under the *Tab* section are named "tab", the names of the extra "floating" objects will be *"**floating-**tab-normal"*, *"**floating-**tab-normal-left"*, … , *"**floating-**tab-toggled"*, *"**floating-**tab-toggled-left"*, etc.

This feature is optional, of course. If Kvantum finds an ***interior*** object for the normal floating tab (*"floating-tab-normal"* in the above example), it will use the floating objects for drawing tabs when there is no tab widget or when it is in the document mode; otherwise, it will use ordinary tabs in all places.

Joined Tabs and Tab Separators

The value of the key *joined_inactive_tabs* under the *General* section is true by default, which means that inactive tabs are drawn joined together, i.e. their right and left frames are not drawn between them. Of course, if that key is set to false, inactive tabs will be drawn separated from each other.

Some themes may need the inactive tabs to be drawn joined together in the "document mode" but not in the ordinary mode, or conversely. That – among other things – is possible with "tab separators".

To simplify the following explanation, we suppose that the value of the key *no_active_tab_separator* (under the *General* section) is *false,* which is its default value. Later we will return to this special key and see how it interacts with tab separators.

When *joined_inactive_tabs* is true, Kvantum looks for "tab separator" objects in the SVG image and only if it finds them, it will draw them between all tabs – inactive and active – properly. The base name of the tab separator objects is made by adding **"-separator"** to the end of the tab frame name under the *Tab* section. They consist of main, top and bottom objects and have "normal" and "toggled" states. If floating tabs are used for the "document mode", they should have their own separator, whose base name is made by adding the string **"floating-"** to the beginning of the ordinary tab separator name.

For example, when the name of the tab frame element is *"tab"*, the name of the ordinary tab separator objects in the normal state are *"tab-separator-normal"*, *"tab-separator-normal-top"* and *"tab-*

separator-normal-bottom", and the name of the floating tab separator objects in the toggled state are *"floating-tab-separator-toggled"*, *"floating-tab-separator-toggled-top"* and *"floating-tab-separator-toggled-bottom"*. Of course, depending on your specific theme, you could omit the top and/or bottom objects but the absence of the main (middle) object means no separator at all.

Therefore, if you want joined inactive tabs in the ordinary mode but separated tabs in the "document mode", you should set *joined_inactive_tabs* to true in the configuration file of your theme, use floating tabs, and add floating separator objects but *not* ordinary ones to your SVG image. Conversely, to have joined tabs only in the "document mode", you should set *joined_inactive_tabs* to true again, use floating tabs, and add ordinary separator objects but *not* floating ones to your SVG mage.

There is also another key that adds more flexibility to how tab separators are drawn, namely *no_active_tab_separator*. Its value of is *false* by default, as we supposed earlier. But if it is set to *true*, tab separators will be drawn only between inactive tabs.

Tab separators can be used for other purposes too. For example, if you want to remove all right and left frames between adjacent tabs, whether they are inactive or active, then, you could set *joined_inactive_tabs* to true (while leaving *no_active_tab_separator* to its default *false* value) and add an invisible rectangle with the name *"tab-separator-normal"* to the SVG image.

The tab separator objects should be drawn for top horizontal tabs – Kvantum will automatically rotate them for other kinds of tabs. Their width is that of tab's right frame and the heights of the top and bottom objects are the heights of tab's top and bottom frames, respectively.

Translucency and Shadow for Menus and Tooltips

If compositing is enabled, menus and tooltips can be translucent and/or have shadow. The following explanation is for menus but it applies to tooltips as well.

Let us suppose that the name of the menu element is "menu", as is the case with the default theme. So, the names of its corresponding SVG objects are *menu-normal*, *menu-normal-top*, *menu-normal-topleft*, *menu-normal-left*, etc. If these nine objects have translucency, the menus will be translucent when compositing is available.

To have shadow, we should set the key *menu_shadow_depth* to a positive value and also add another group of frame objects, whose names include the word "shadow" as the second word in their names, i.e. *menu-**shadow**-top*, *menu-**shadow**-topleft*, *menu-**shadow**-left*, etc. The shadow and the menu frame should together be divided into these eight objects. For example, *menu-**shadow**-top* draws the shadow for the top part of menus and when menus have frame, it also includes the top part of their frame. The object *menu-normal* is used, with or without shadow, for drawing the interior of menus.

The keys *menu_shadow_depth* and *tooltip_shadow_depth*, in the General section of the configuration file, control the width of menu and tooltip shadows respectively.

If the key *composite* is set to false or the environment does not support compositing, menus and tooltips will be drawn without translucency and shadow.

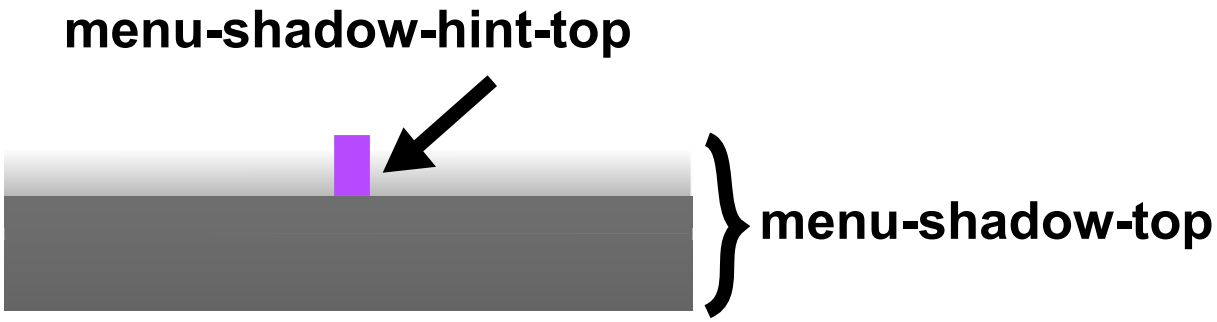
Blurring for Menus and Tooltips:
“Shadow Hint” Rectangles

If (pop-up) blurring is enabled (which is possible only under KDE), the regions behind translucent menus and tooltips will be blurred. But for blurring not to include their shadows, extra “shadow-hint” rectangles should be appropriately drawn to inform Kvantum about pure shadows.

For example, suppose that the name of the menu element is “menu”. Then, there are four “shadow-hint” rectangles with these names:

*menu-**shadow-hint**-top*
*menu-**shadow-hint**-bottom*
*menu-**shadow-hint**-left*
*menu-**shadow-hint**-right*

They determine the height or width of the *purely shadowy* parts of the top, bottom, left and right frames respectively. For example, the height of *menu-**shadow-hint**-top* is that of *menu-**shadow**-top* minus the height of the top frame included in it, etc. The following image shows this:



Therefore, only the heights of *menu-shadow-hint-top* and *menu-shadow-hint-bottom* are important, while for *menu-shadow-hint-left* and *menu-shadow-hint-right*, only the widths are pertinent.

The same is true for tooltips, of course. So, there can be eight “shadow-hint” rectangles in total.

Even if you do not use blurring with your theme, add these eight rectangles when your menu and tooltip objects have shadow because, on the one hand, the user might enable blurring with **Kvantum Manager** later and, on the other hand, they are used in positioning menus.

* * *

Kvantum announces its real menu shadow sizes with the *QObject* property “*menu_shadow”*, which can be retrieved in any application by a line like the following:

widget→style()→property("menu_shadow").value<QList<int>>();

The retrieved *QList* contains shadow thicknesses in this order: **left → top → right → bottom**.

Window Translucency

Whole windows and dialogs can be made translucent. That needs compositing, a true value for *translucent_windows* in the configuration file and a translucent SVG image for the interior element of the Window section. As is the case with menus and tooltips, there will be no translucency if compositing is not enabled in the configuration file or not supported by the environment.

Some applications are not compatible with window translucency and may show totally transparent windows or even crash. They are usually Qt video playing applications (although some Qt-based video players, like VLC, support window translucency). You could exclude them by adding the names of their executable files to the *opaque* key.

Maximum Corner Roundness and Frame Expansion

Although the four SVG objects used for drawing corner frames (-topleft, -topright, -bottomleft, -bottomright) can be quite curved, the degree of corner roundness depends on the frame widths too and so, it cannot be high.

However, Kvantum has a key that can expand frames, namely ***"frame.expansion"***. If its value is greater than zero under any widget section, the frames of that widget will be expanded until the corner frames meet each other either horizontally or vertically, depending on the aspect ratio of the actual widget, *provided that at least the height or the width of the actual widget is less than or equal to the value of "frame.expansion"*. The expansion is so that the corner frame objects become equal squares even when they are not drawn as SVG squares.

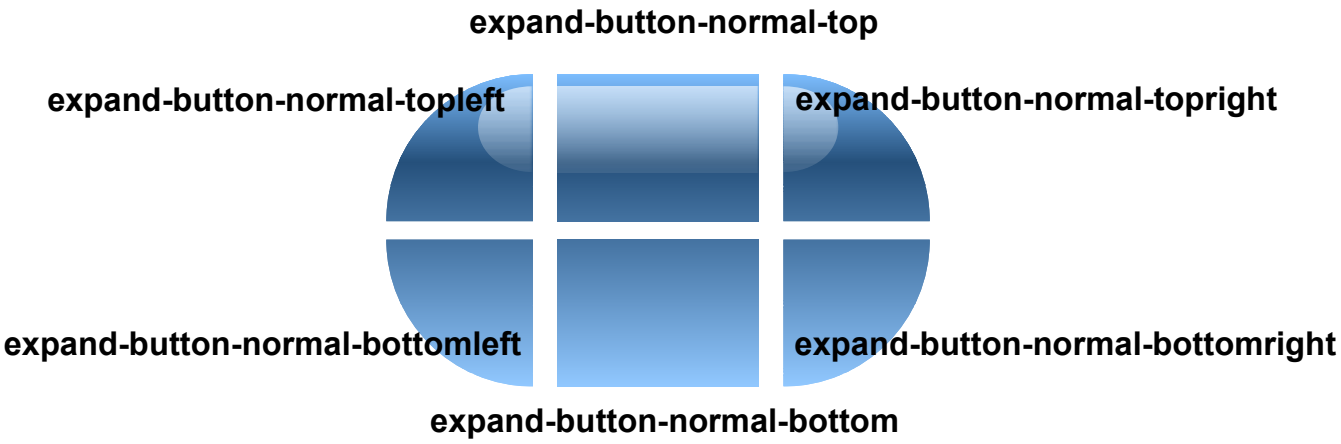
You could use this key to make widget corners as rounded as possible. By giving a positive value to *"frame.expansion"*, you could not only decide which widgets have extremely rounded corners but also set size limits, beyond which, they should have ordinary corners (because too big widgets would look weird with completely rounded corners).

The SVG objects used for drawing the completely rounded corners can be the usual ones, in which case you would not need to add anything to your SVG image. But, except for totally flat objects (which do not have color gradient), you might want to add extra objects for maximally rounded widgets. If so, you should name them by adding the prefix ***"expand-"*** to the beginning of the usual object names. For example, for maximally rounded buttons with the frame element named as "button" under the *PanelButtonCommand* section, the names of the objects that are used specifically for complete rounding are *"expand-button-normal-top"*, *"expand-button-normal-topleft"*, *"expand-button-normal-topright"*, …, *"expand-button-focused-top"*, *"expand-button-focused-topleft"*, *"expand-button-focused-topright"*, etc. The interior, left or right objects are not used in drawing and you do not need to include them. However, Kvantum looks for the ***top objects*** (whose names end with "-top") and only if it founds them, it will use the other *"expand-"* objects. Of course, you should draw the corner objects rounded and give all objects appropriate color gradients if any.

In case of *"expand-"* objects, the color gradients of opposite frame objects should be complementary,

so that when they are adjacent to each other due to frame expansion, a smooth gradient forms.

The following image can serve as an example:



As you can see, in the above image, there are no objects for the interior, the right frame or the left frame. It is obvious why the interior object is redundant. As for the right and left frames, if the height of widget is greater than its width, Kvantum first rotates its rectangle by 90 degrees, draws it by using the available objects and then rotates it by 90 degrees again but in the opposite direction, so that right and left objects are not needed (this is only a rough description). The reason is that the widget looks more natural in this way.

Although the interior object (alongside the left and right ones) is not used in this, *if it exists and if its base name is identical to the frame element name*, the corners of those widgets, whose heights and widths are greater than the value of *"frame.expansion"*, will also be rounded but not maximally. This can be called "partial frame expansion".

Kvantum looks for the ***"expand-" interior object*** to do partial frame expansion. If, in addition, it finds the ***"expand-" top frame object***, it will only use the *"expand-"* objects for partial frame expansion; otherwise, the usual objects will be used.

This feature is especially useful when you want rounder corners with the *usual* objects but without increasing the frame thickness. For that purpose, you could just add a black rectangle for the *"expand-"* interior object to inform Kvantum that you want partial expansion (the color is not important). Then, Kvantum will use your usual frame objects for partial frame expansion, according to the value of the key *"frame.expansion"* in the configuration file.

The theme "KvSimplicity", which is installed among the extra themes, contains an example of the above-mentioned feature. Its buttons have 3-px thick frames but their corner roundness is what 5-px frames could afford (frame.expansion=10).

Of course, all *"expand-"* objects can also be added to the SVG image, such that they are used, instead of the usual objects, for partial frame expansion. Anyway, even if you add all *"expand-"* objects, never remove the usual objects because they will be needed in some places!

The key *"frame.expansion"* can also be used for purposes other than corner rounding. The main idea is to make the corner objects as big as possible so that the left and right (or top and bottom) sides of

widgets get their shapes.

Please also note that:

(a) Sliders, scrollbars, header sections and container widgets (menus and tab widgets, for example, but not menu-items or tabs) do not support frame expansion. You could still round the edges of sliders and scrollbars by not giving them any interior, drawing appropriate images for their corner frames, and choosing their left and right frame widths equal to half of their widths. If you do so, it is better that you also set their widths to even numbers. However, tab widgets (under the section *TabFrame*) whose tabs are not attached to them (*attach_active_tab=false*) and also group-boxes whose labels are above their frames (*groupbox_top_label=true*) can have frame expansion,

(b) Item-views can only be *partially* rounded and so, their *"expand-"* interior object should be present for frame expansion.

(c) Some instances of other widgets may fall back to the usual SVG objects under certain conditions (lack of space, for example).

* * *

Frame expansion may seem complex at first but it can be summarized as follows:

1. With all *"expand-"* frame objects present, if the widget height is less than *"frame.expansion"*, the corners will be completely rounded.

2. With the *"expand-"* interior object, if the widget height is greater than *"frame.expansion"* and the interior element has the base name of the frame element, the frame will be partially rounded. In this case, if the *"expand-"* top frame object is missing, the usual objects will be used for frame expansion.

Frame Expansion and Border

Since, in the case of frame expansion, the frames themselves are expanded, they cannot serve as a border anymore. When the *"expand-"* objects have gradient, there is no need to a border, although widgets may look more elegant with it. But, when they are flat, a border may be really needed. However, there is a way to give the completely (or partially) rounded widgets a nice border. That is done by using the ***"border-"*** objects. They are exactly like the *"expand-"* objects except for their names, which are started with the string *"border-",* and their background color, which is the color of the desired border. Although their shapes should be identical to the shapes of their corresponding *"expand-"* objects, Kvantum uses them to make a border by making the *"expand-"* objects a little smaller and putting them inside the *"border-"* objects. The thickness of the resulting border is equal to the expanded frame width – or to the ordinary frame width when there is no expanded frame size in the configuration file (see the explanation of *frame.expanded.top*,...,*frame.expanded.right* in the sections table of *Theme-Config.pdf*).

Again, if the interior, left and right *"border-"* objects also exist alongside the interior, left and right *"expand-"* objects, not only the corners of those widgets, whose heights and widths are greater than the value of *"frame.expansion",* will be partially rounded, but also they will have border.

The themes “KvCurves”, “KvCurves3d” and “KvCurvesLight”, which are installed among the extra themes, are good examples for frame expansion with border.

Dealing with exceptions

Before Kvantum was created, many developers wrongly presupposed fixed values for some *QStyle* parameters and used them in the hard-coded styles of their applications. This is still the case. For example, the frame thickness may be considered to be only 3px, although it can have different sizes for different kinds of widgets and even on the top, bottom, right and left of the same widget.

A hard-coded style will be correct only if it is complete and does not include any false presupposition about *QStyle* parameters. For example, if it sets the foreground color, it should also set the background color; otherwise, some texts will not be readable with some styles. Unfortunately, there are many exceptions in this regard and some developers do not use *QStyle* correctly in their applications.

Kvantum deals with such exceptional cases as far as possible. For example, if the foreground is set but it is not readable, Kvantum might correct it or set an appropriate background. Or if there is not enough space for drawing the text of a button, Kvantum will set the frame widths to 3px and the text margins to 2px for it, although those values may be greater in the active Kvantum theme.

However, Kvantum could not deal with all exceptional cases if the theme is not ready for them. There are simple rules that can make most applications with wrong hard-coded styles look good enough:

(1) Under the *ItemView* section, set *text.press.color* and *text.toggle.colo*r in such a way that they do ***not*** have a high contrast with *highlighted.text.color* under the *GeneralColors* section. The reason is that it may be supposed that all of these colors are the same. Although that is not enforced by *QStyle*, some hard-coded styles may need it.

Also, set *text.normal.color* and *text.focus.color* for *ItemView* in such a way that they do not have a high contrast with *text.color* under *GeneralColors* and use a translucent (semi-transparent) background for the focused interior SVG element of *ItemView* in the SVG file.

If you use inactive colors in your theme, set *text.toggle.inactive.color* for *ItemView* to some value near *inactive.highlighted.text.color* under *GeneralColors* (both keys should be defined if inactiveness is used) and add appropriate SVG elements for toggled-inactive *ItemView* to the SVG file.

(2) If you add inactive elements to the SVG file, be sure to set *inactive.highlight.color* to a value different from *highlight.color* under the *GeneralColors* section of the kvconfig file; otherwise, Qt will not update inactive widgets correctly (a Qt bug?).

(3) Include SVG elements that look good with a 3px frame thickness. If you use frame expansion, add non-expanded SVG elements too because Kvantum may disable frame expansion for some widgets. If you use the usual SVG elements with frame expansion and if they are not suitable with a 3px frame thickness, you could use *frame.expandedElement* in addition to *frame.element* and give a 3px frame to the latter. You could also use *frame.expanded.top/bottom…* in addition to *frame.top/bottom…*.

(4) Always use an opaque color for *base.color* (under *GeneralColors*); otherwise, item-views of some applications might show what is "behind" them (because of a bug in those applications).

(5) Do not forget to add appropriate flat indicators (arrows). They are also needed with good hard-coded styles when *highlight.color* – under *GeneralColors* – has a high contrast with *base.color*.

Theme Installation Paths

The default user installation path, which Kvantum Manager uses, is always *'~/.config/Kvantum/$THEME_NAME/'*, and the default root installation path, used by Kvantum's extra themes, is *'$DATADIR/Kvantum/$THEME_NAME/'* ($DATADIR is often '/usr/share' but depends on the Kvantum installation prefix).

To make theme packaging easier, three extra installation paths are (unwillingly) added too, namely, *'~/.themes/$THEME_NAME/Kvantum/'*, *'~/.local/share/themes/$THEME_NAME/Kvantum/'* and *'$DATADIR/themes/$THEME_NAME/Kvantum/'*.

Kvantum uses the concept of ***priority*** for theme installation. If the same theme is installed in more than one path, the one whose path has the highest priority will be used. The user paths always take priority over the root ones. All paths, arranged in the order of their priorities from high to low, are as follows:

~/.config/Kvantum/$THEME_NAME/
~/.themes/$THEME_NAME/Kvantum/
~/.local/share/themes/$THEME_NAME/Kvantum/
$DATADIR/Kvantum/$THEME_NAME/
$DATADIR/themes/$THEME_NAME/Kvantum/

Because of this hierarchy, if you manually remove a Kvantum theme, it might still be shown on the list of installed themes by Kvantum Manager. Also if you manually install a newer version of a theme, an older version with a higher priority might still be used. However, if you install a theme with Kvantum Manager, it will always take priority over its other installations.

Therefore, in the case of theme installation or updating *without* Kvantum Manager, it is better first to delete the same theme with Kvantum Manager if it is already installed because Kvantum Manager takes into account all user installation paths when deleting a theme.

A Word about KDE's Symbolic Icons

Kvantum recognizes KDE's symbolic icons, which change color to have enough contrast with their backgrounds, but since Kvantum is not limited to any desktop environment, it can have themes that are not compatible with KDE symbolic icons. Therefore, if you want your dark-and light theme to be compatible with them, you should choose *highlight.text.color*, under the *GeneralColors* section, in such a way that it has a high contrast with *window.text.color*. Of course, you should also choose an appropriate value for *highlight.color*.

Dark variants of themes

Some light themes may come with their dark variants. For example, *KvSimplicity* (which is included in Kvantum) has a dark variant called *KvSimplicityDark*. If a theme has a dark variant, it will be better to name the latter by adding the suffix ***"Dark"*** to the name of the former and also set the key *dark_titlebar* to *true* for it under the General section. The reason is threefold: (1) The user could find the dark variant of a light theme more easily; (2) Qt applications could use the dark variant of an active light theme when, for example, they are started with the command-line option *"-style kvantum-dark"*; and (3) under GTK environments like Gnome, the title-bar will be dark for dark variants automatically.

Instead of being separate themes, dark variants could be included in their corresponding light themes if their names end with ***"Dark"***, so that a light theme folder with the name *"My_Theme"* may contain all of these files:

My_Theme.svg (the light SVG image)
My_Theme.kvconfig (the light Kvantum configuration file)
My_ThemeDark.svg (the dark SVG image)
My_ThemeDark.kvconfig (the dark Kvantum configuration file)

The dark variants that are separate themes with their own theme folders take priority over those included in their corresponding light folders, so that if *"My_ThemeDark"* is both a separate theme and included in *"My_Theme"*, only the former will be picked up by Kvantum.

* * *

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 configuration 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.