# QSvgStyle Documentation

Saïd Lankri, said.lankri@gmail.com

May 23, 2014

# 1 What is QSvgStyle ?

QuantumStyle (QS) is an **SVG themable** style for Qt and KDE applications.

- **Themable** means that the style uses **themes** to draw widgets. A widget is a graphical element, like a push button or a combo box. To draw a widget, the theme looks for the corresponding element in the theme. You can see a theme as a collection of images, one image is used for push buttons, another for menus, ... The idea behind using themes is that one can get different application looks by choosing a theme and without changing the style. One important difference between themable styles and non themable styles is that themes can be created by anyone without knowledge of C++ programming, as creating a theme is just a matter of drawing widget elements and some design.

- **SVG** means that themes, that contain elements used to draw graphical widgets, are stored in the form of SVG files. An SVG file is a particular type of picture in which you can give names to objets among other things. SVG files can be created using the open source vector drawing program **Inkscape** (`http://www.inkscape.org`). In order to use a theme, each SVG file is associated with a **theme configuration** file. The SVG file stores the graphical elements, and the configuration file tells how to use these elements for widgets.

# 2 Features

QS has the following features :

- **Default theme**. QS has a built-in default theme that is used when the user has not set any specific theme to use.

- **User set theme**. The user can instruct QS to use some particular theme. The selected theme will be used to draw all widgets of all applications.

- **Per application theme**. In addition to the default and user set themes, QS can also draw applications with specific themes. This means that you can use a different theme for each application.
- **Theme Builder**. To help the writing of **theme configuration** files, QS comes with a small utility called **Theme Builder**. This user friendly tool helps theme designers to easily create and modify theme configuration files.
- **The Debug Mode**. To help theme designers develop their themes, QS features a **debug mode**. In this mode, in addition to drawing widgets, QS adds rectangles with specific colors for various parts of the widget. Note that debug mode can only be activated at compilation time.

# 3 How are widgets drawn ?

⇒ When drawing a widget, the themes looks for the corresponding elements in the following order :

1. in the **per application theme** if it exists
2. in the **user set theme** if it is set and exists
3. in the **default QS theme** which is built-in

⇒ The application name is guessed by calling `QApplication::applicationName()`

# 4 Understanding widget drawing

From QS perspective, **all** widgets are drawn inside their **bounding rect**. They may have a **frame**, an **interior**, **label** specifications and **indicator** specifications. In addition, some size related specifications can be set, for instance one can force a minimal or a fixed size for a widget.

⇒ Not all widgets make use of these settings. Basically, **all** widgets accept the **frame** and **interior** settings. Widgets that can display text or icons accept the **label** settings. Widgets that can display indicators (e.g. tool buttons that can display drop down arrows) accept the **indicator** settings.

## 4.1 Bounding Box

The bounding box of a widget is the area where the entire widget is drawn.

## 4.2 Frame

All widgets can have a **frame**. The frame is drawn immediately inside the bounding box with no margins. The **width** (in pixels) of the frame can be adjusted for each of the four sides (top, bottom, left, right). The rect inside the frame is called the **frame rect**.

## 4.3 Interior

All widgets can have an **interior**. The interior is drawn inside the **frame rect** and can have a **margin**. The rect inside the interior in called the **interior rect**.

## 4.4 Label

Widgets that can display text, an icon or both have a **label**. The label is drawn inside the **interior rect** and can have a **margin**. The rect inside the label is called the **label rect**. The spacing between the icon and the text is called the **text-icon spacing**.

## 4.5 Indicator

Widgets that can display indicators have an **indicator** setting. The indicator is drawn inside the **interior rect** with the same margins as the label and the same spacing.

# 5 Widget "status"

QS can draw a widget differently based on its status. There are five supported statuses :

- "normal". The widget is in a normal status. All but the below.
- "focused". The mouse is over the widget. NOTE : the term is misleading, as we should use "hoovered" or "highlighted".
- "pressed". The widget is pressed.
- "toggled". For widgets that can be toggled (e.g. tool buttons), the widget is toggled.
- "disabled". The widget is disabled.

**Both** the frame and the interior are drawn based on these statuses (e.g. a "pressed" widget will be drawn with the "pressed" frame and the "pressed" interior).

# 6 SVG theme design

Understood widget drawing principles and ready to make your own theme ? Let's start with an exemple. We want to design a push button.

## 6.1 Interior example

First, we need an SVG file. Let's create one with **Inkscape** and call it `myTheme.svg` This SVG file will contain the **frame** and **interior** elements of our push buttons. Remember that for each widget, QS supports five statuses, so we need five elements for the frame, and five elements for the interior.

To let QS correctly extract these elements to draw a widget, each element must have a **name** which obey certain rules. Of course, one can put any other object in the SVG file. Object which do not obey naming rules or carry unrecongnized names will be ignored by QS.

So let's start by the **interior**. We need five interiors, like this :

**Advice**

- An interior can consist of **several objects**. To make QS able to use such an interior, you have to **group** these objects into one object. Object grouping can be achieved by selecting the desired objects and pressing `CTRL+G`.

- Even if your interior consists only of one object, it is a good idea to group it.

- In order to "work" inside a group, it is not necessary to ungroup. Just **double click** on the object group to **enter** the group. You'll now be able to work on indivial objects inside the group. To **exit** the group, you have to click on an object which does not belong to the group you're working on.

## 6.2 SVG Naming Conventions

Each of the five statuses must obey some naming conventions. for an interior, QS expects a name in the form `<base>-<status>`.

- `base` is a basename for the interior that you can set freely. For example, as we are drawing a push button interior, we can set it to `button`.

- `status` is the status of the element. There are five possibilities : `normal`, `focused`, `pressed`, `toggled`, and `disabled`.

For example, the element which will be used by QS to draw a normal push button interior has the name `button-normal`.

To **assign** a name to an element, press `CTRL+SHIFT+o`. A dialog box will appear. Type in the **Id** field the name of the element and don't forget to **press enter** or click the "set" button.

### 6.3 Frame example

As for interiors, we need five frames. But it is a little more complicated. Each frame consist of eight elements named `<base>-<side>-<status>`. Where `<base>` is a freely chosen basename, `<side>` is one of `top`, `bottom`, `left`, `right`, `topleft`, `topright`, `bottomleft`, `bottomright`, and `<status>` is one of the five statuses supported by QS.

It is not possible to simply group these elements as for interiors, because contrary to interiors, frames must not be stretched when drawn.

### 6.4 Continuing

You can apply these steps to draw other elements for widgets. Please note the following :

- You do not need to draw elements for each type of widget. **The same elements** can be reused for multiple widgets. This is specified later in the theme configuration file.

- Some widgets have special naming conventions. See the default theme for an example.

## 7 Writing a Theme Configuration File

Once you have designed the svg file, we need a **theme configuration** file. If your SVG file is `myTheme.svg` then the theme configuration file must be named `myTheme.qsconfig`.

Theme configuration files are created using the QS theme builder. Launch it and press "New" button. Give the name `myTheme.qsconfig` and start playing with the settings. Note that the settings of the default theme are copied to the newly created theme. On the left you select the widget you to adjust the settings for, on the right you set the frame, interior, text and indicator settings. Most important part the the **basename** setting. This is the freely chosen name above (without status for interior, without status and side for frame).

# 8 Make QS aware of your theme

To make QS aware of your theme, some steps are necessary :

- Put both `myTheme.svg` and `myTheme.qsconfig` in a directory called `myTheme`
- Put that directory under `$HOME/.config/QuantumStyle/`

To make QS use yout theme, put the line

```
theme=myTheme
```

in the file `$HOME/.config/QuantumStyle/quantumstyle.qsconfig`. This will instruct QS to use the theme `myTheme` for **all** applications.

# 9 Per application themes

Making QS aware of a per application theme follows the same procedure as above, except for the line in `quantumstyle.qsconfig`.

If your application name is `konqueror` for example, there must be a `konqueror` directory in `$HOME/.config/QuantumStyle/` which contains both `konqueror.svg` and `konqueror.qsconfig`.

# 10 Notes

To avoid any inconsistency, please provide both the SVG and the theme configuration file.
Providing the SVG without the theme configuration file will lead QS to draw the theme elements using the default configuration settings.
Providing the theme configuration file without the SVG will lead QS to draw the default elements with the settings of the theme.