**使用Qt Style Sheets制作UI特效**

Jump to: [navigation](http://www.developer.nokia.com/Community/Wiki/%e4%bd%bf%e7%94%a8Qt_Style_Sheets%e5%88%b6%e4%bd%9cUI%e7%89%b9%e6%95%88#mw-head), [search](http://www.developer.nokia.com/Community/Wiki/%e4%bd%bf%e7%94%a8Qt_Style_Sheets%e5%88%b6%e4%bd%9cUI%e7%89%b9%e6%95%88#p-search)

引言

作为一套GUI框架，Qt是非常强大的。（注：Qt 不仅是一套优秀的GUI框架，同时也是一套出色的应用程序框架）。  
在UI的制作方面Qt为广大开发者提供了一套强大而易用的工具，她就是——Qt Style Sheets。  
本文将向大家举例介绍如何使用Qt Style Sheets制作个性化的UI界面。例子程序（stylesheetDemo）可通过本文末尾所附链接下载。

UI涉及的东西非常庞杂，Qt Style Sheets也包含许许多多的内容，因此本文并不试图对Qt Style Sheets进行系统的理论性的详解，那需要数十倍于本文的篇幅。本文仅通过几个例子，将大家引入Qt Style Sheets的大门，以后如有更多需求大家直接在Qt Assistant中查询Qt Style Sheets并且结合自己写的程序进行测试就可以了。

测试设备

Nokia N8

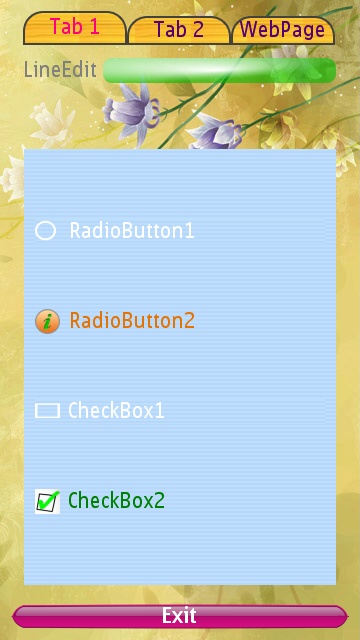
预备知识

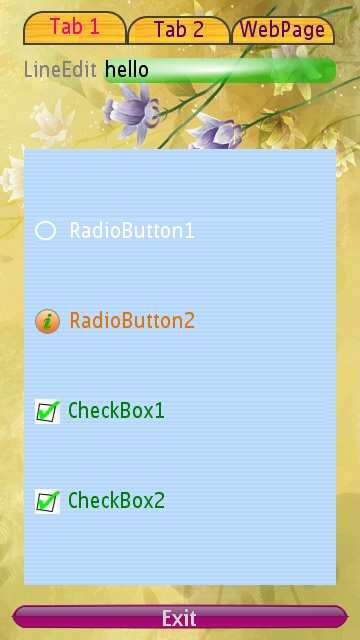
Style sheets 是由一系列的style rules组成的。一条style rule 由选择器selector和声明declaration这两部分构成。selector说明这条规则在哪些widgets上起作用，declaration说明要在这些widgets上设置什么属性properties。例如：

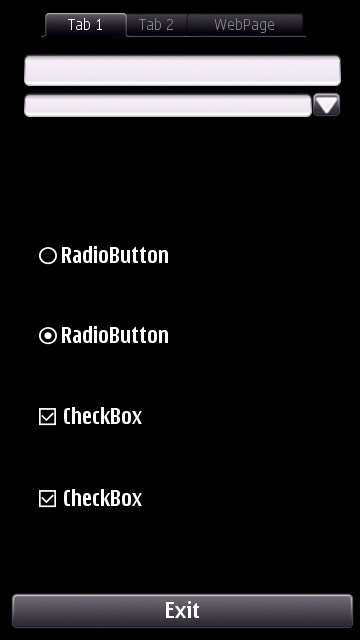
QPushButton, QLineEdit { color: red; background-color: white }

在上面这条style rule中QPushButton, QLineEdit 是两个选择器，中间用逗号连接。 { color: red; background-color: white }是声明declaration，大括号里面是一系列的 property: value对，中间用分号连接。这条规则指出对QPushButton和QLineEdit 以及他们的子类需要使用红色作为其前景色，并使用白色作为其背景色。  
Qt widgets所支持的所有属性列表请查阅[List of Properties](http://doc.qt.nokia.com/qt-maemo-4.6/stylesheet-reference.html#list-of-properties)

Tab1：QLineEdit QGroupBox QRadioButton QCheckBox QLabel（使用qss文件）

例子程序的UI结构非常简单，只有两部分，上方是一个有三个tab页面的QTabWidget，下面是一个QPushButton。  
下面我们先来制作TabWidget的第一个页面Tab1。先看一下效果图：  
图一：  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab1_1.jpg)

图二：  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab1_2.jpg)

这张是没有使用StyleSheet的样子：  
[](http://www.developer.nokia.com/Community/Wiki/File:1.jpg)

Tab1中使用到了五种控件。如果控件较多或比较复杂，我们可以通过使用qss文件来设置Style Sheet。首先我们新建一个文本文档，后缀名改为qss，然后用文本编辑器比如记事本打开它，将我们设置的Style Sheets写进去然后保存就可以了。本例程创建的qss文件叫stylesheetDemo.qss，于是我们在程序中只需要写如下几行代码就可以使我们写在qss文件中的Style Sheets起作用：

QFile file(":/qss/stylesheetDemo.qss");  
file.open(QFile::ReadOnly);  
QTextStream filetext(&file);  
QString stylesheet = filetext.readAll();  
ui->tab->setStyleSheet(stylesheet);

程序中stylesheetDemo.qss已加入到资源文件，其中ui->tab就是TabWidget中的第一个tab页面。

下面是stylesheetDemo.qss的内容：

QGroupBox {  
 background-image: url(:/pics/background.png);   
 border-radius: 30px;  
}  
   
QLabel {  
 color: gray;  
}  
   
QLineEdit {  
 background: qradialgradient(cx:0, cy:0, radius: 1,  
 fx:0.5, fy:0.5, stop:0 white, stop:1 rgba(0,190,0, 60%));  
 border-radius: 9px;  
}  
   
   
   
QCheckBox:checked {  
 color: green;   
}  
   
QCheckBox::indicator {  
 width: 25px;  
 height: 25px;  
}  
   
QCheckBox::indicator:checked {  
 image: url(:/pics/checkbox.gif);  
}  
   
   
   
QRadioButton{  
 spacing: 10  
}  
   
QRadioButton::indicator {  
 width: 25px;  
 height: 25px;  
}  
   
QRadioButton:checked {  
 color: rgb(230,115, 0);   
}  
   
QRadioButton::indicator:checked {  
 image: url(:/pics/radioButton.png);  
}

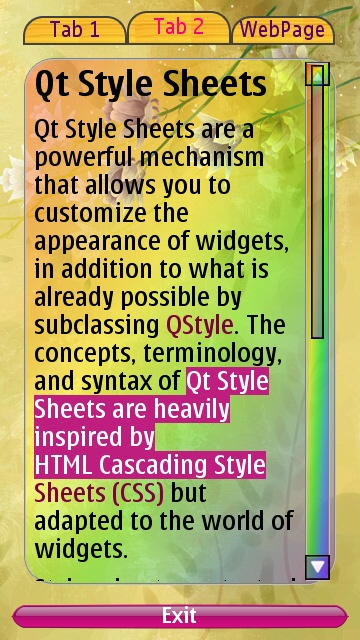
其中border-radius指的是边框四角的半径，这一属性可以制作出弧形的边框。  
  
background-image属性设置控件的背景图片。  
background-color 设置控件的背景色，我们这里对QLineEdit使用了渐变的颜色，这里利用了Qt提供的qradialgradient  
一个冒号说明的是状态，例如“:checked”指的是当此控件被checked的时候。  
双冒号说明的是子控件，例如“::indicator”指的是 QCheckBox、QRadioButton、QAbstractItemView 或者是可以被选中的 QMenu item或QGroupBox的indicator。

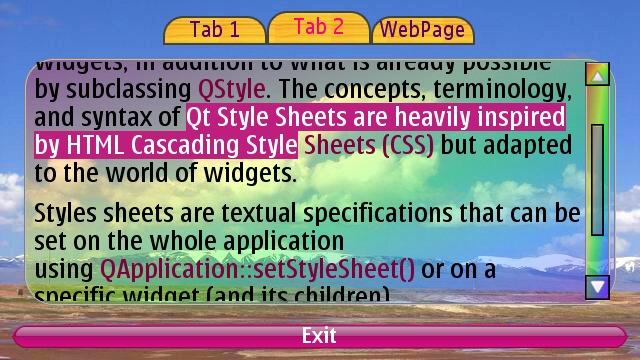
这里需要注意的是，由于QRadioButton和QCheckBox在Symbian上的实现有一点缺憾，就是他们在获得焦点的时候，其新的背景颜色会完全覆盖掉控件，用户就看不到控件了。因此我们需要去掉他们获得焦点的能力：

ui->checkBox->setFocusPolicy(Qt::NoFocus);  
ui->checkBox\_2->setFocusPolicy(Qt::NoFocus);  
ui->radioButton->setFocusPolicy(Qt::NoFocus);  
ui->radioButton\_2->setFocusPolicy(Qt::NoFocus);

Tab2：QTextBrowser （在代码中setStyleSheet）

程序中对TextBrowser设置了一种透明的背景颜色，并且是像彩虹一样逐渐变化的颜色。这主要是利用了qlineargradient。下面分别是竖屏和横屏状态下Tab2的效果图：

图三：  
  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab2_1.jpg)

图四：  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab2_2.jpg)

这张是没有使用StyleSheet的样子：  
[](http://www.developer.nokia.com/Community/Wiki/File:2.jpg)

我们这里直接在代码中对textBrowser设置StyleSheet：

ui->textBrowser->setStyleSheet("**\** color: rgb(127, 0, 63);**\** background-color: qlineargradient(x1: 0, y1: 0, x2: 1, y2: 1, **\** stop: 0 rgba(255, 0, 0, 30%), stop: 0.2 rgba(255, 128, 0, 30%), stop: 0.4 rgba(255, 255, 0, 30%), **\** stop: 0.6 rgba(0, 255, 0, 30%), stop: 0.8 rgba(0, 128, 255, 30%), stop: 1 rgba(128, 0, 255, 30%)); **\** selection-color: white;**\** selection-background-color: rgb(191, 31, 127);**\** border: 2px groove gray;**\** border-radius: 30px;**\** padding: 2px 4px;");

Tab3：QWebView

QWebView也是可以通过Qt Style Sheets的方式在**一定程度上**修改网页呈现在用户面前的样子。  
例程中对WebView设置了完全透明的背景色，下面是效果图：  
图五：  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab3_1.jpg)

图六：  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab3_2.jpg)

图七：  
[](http://www.developer.nokia.com/Community/Wiki/File:Tab3_3.jpg)

这张是没有使用StyleSheet的样子：  
[](http://www.developer.nokia.com/Community/Wiki/File:3.jpg)

ui->webView->setStyleSheet("border: 1px groove gray; border-radius: 5px; background-color: rgba(255, 193, 245, 0%); ");

这里要注意，这样设置只对本身透明的网页是有效的，如果网页自己设置了白色背景，则我们还是看不到透明的效果。  
  
还要额外说明一点，如果不对webView的border属性进行设置，而使用QWebView在N8上的默认实现，则网页中的Button是黑色的背景，Button上的字是看不清的。  
要想完全使网页按照我们自定义的样式进行显示（渲染），最根本的解决办法是我们修改Webkit，从而渲染出我们需要的样子。

QPushButton QTabWidget

对比图一和图二，我们会发现exit按钮按下和没有按下时的背景、文字颜色和文字位置都是不一样的，其中背景是通过border-image实现的，文字的位置是通过padding来控制的。

ui->ExitpushButton->setStyleSheet("**\** QPushButton {**\** color: white;**\** border-image: url(:/pics/button.png);**\** border-width: 12px;**\** padding: -12px 0px;**\** min-height: 25px;**\** min-width: 60px;**\** }**\** QPushButton:pressed {**\** color: lightgray;**\** border-image: url(:/pics/button-pressed.png); **\** padding-top: -10px;**\** padding-bottom: -16px;**\** }**\** ");

对于三个tab标签的样式是这样设置的，其中!selected表示没有选中，margin-top: 5px;会使得选中的tab比没选中的高5个像素。

ui->tabWidget->setStyleSheet("**\** QTabBar::tab {**\** color: rgb(84,2,119);**\** background-image: url(:/pics/wood.jpg); **\** border: 2px solid rgb(68,66,64);**\** border-bottom-color: rgb(68,66,64); **\** border-top-left-radius: 20px;**\** border-top-right-radius: 20px;**\** max-height: 21px;**\** min-width: 8ex;**\** padding: 2px;**\** } **\** QTabWidget::tab-bar {**\** alignment: center;**\** } **\** QTabBar::tab:!selected {**\** margin-top: 5px; **\** }**\** QTabBar::tab:selected {**\** color: rgb(255,0,128); **\** }**\** ");

最后横竖屏背景图片的切换也是通过stylesheet实现的：

void MainWindow::resizeEvent ( QResizeEvent \* event )  
{  
 enum ScreenMode currentscreenMode;  
 if(size().height()> size().width())  
 currentscreenMode = Portrait;  
 else  
 currentscreenMode = Landscape;  
   
 if (currentscreenMode!=scmode)  
 {  
 scmode = currentscreenMode;  
 switch(scmode)  
 {  
 case Portrait:  
 this->setStyleSheet("QMainWindow{ background-image: url(:/pics/bgPortrait.jpg)}");  
 break;  
 case Landscape:  
 this->setStyleSheet("QMainWindow{ background-image: url(:/pics/bgLandscape.jpg)}");  
 break;  
 default:  
 break;  
 }  
 }  
}