La coloration syntaxique dans QTextEdit

Qt by Nokia

Par traducteur: charlespf - Qt Quarterly

Date de publication : 21 mars 2012

Dernière mise à jour : 12 avril 2012

L'utilisation de couleurs et de polices appropriées pour mettre en évidence différents éléments de programmation et balises de langages aide le cerveau à comprendre les structures de documents. En signalant les erreurs de syntaxes, la coloration syntaxique interactive aide également à réduire le temps utilisé dans le cycle « compiler, exécuter, tester ». La coloration peut même être utilisée pour signaler les erreurs de langages dans les documents sous forme de texte. Avec Qt, ajouter la coloration syntaxique dans un **QTextEdit** est très simple, c'est ce que cet article va montrer.

Commentez

I - L'article original.....3
II - Introduction.....3
III - Sous-classer QsyntaxHighlighter.....3
IV - Garder les traces des informations d'état.....3
V - Un surligneur de code HTML basique.....5
VI - Résumé.....9
VII - Remerciements.....9

Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

I - L'article original

Qt Quarterly est une revue trimestrielle électronique proposée par Nokia à destination des développeurs et utilisateurs de Qt. Vous pouvez trouver les **versions originales**.

Nokia, Qt, Qt Quarterly et leurs logos sont des marques déposées de *Nokia Corporation* en Finlande et/ou dans les autres pays. Les autres marques déposées sont détenues par leurs propriétaires respectifs.

Cet article est la traduction de l'article **Syntax Highlighting in QTextEdit** paru dans la Qt Quarterly Issue 21.

Cet article est une traduction de l'un des tutoriels en anglais écrits par **Nokia Corporation and/or its subsidiary(-ies)** , inclus dans la documentation de Qt. Les éventuels problèmes résultant d'une mauvaise traduction ne sont pas imputables à Nokia.

II - Introduction

Qt 3.1 introduit la classe abstraite **QSyntaxHighlighter** pour fournir la coloration syntaxique à une classe **QTextEdit**. Dans Qt 4, cette classe a été déplacée dans la bibliothèque Qt3Support et une nouvelle classe **QSyntaxHighlighter** a trouvé sa place dans Qt 4. Cette nouvelle classe est basée sur le nouveau moteur de texte riche de Qt 4 et fonctionne de façon arbitraire avec des objets **QTextDocument**. Cet article va se concentrer sur cette nouvelle classe.

III - Sous-classer QsyntaxHighlighter

Ajouter la coloration syntaxique sur un objet **QTextEdit** implique de sous-classer **QSyntaxHighlighter**, en réimplémentant la méthode highlightBlock() et en initialisant la sous-classe **QSyntaxHighlighter** avec la classe **QTextEdit** et la classe **QTextDocument** sous-jacente (retournée par **QTextEdit::document()**) comme parent.

QTextDocument appellera ensuite highlightBlock() pour chaque ligne du document si nécessaire. Dans la réimplémentation de highlightBlock(), on peut appeler la méthode setFormat()pour paramétrer le formatage des différents éléments de la ligne. Le bout de code suivant montre une réimplémentation de highlightBlock(), où les caractères qui ne sont pas alphanumériques sont colorés en vert.

```
void MyHighlighter::highlightBlock(const QString &text)
{
    for (int i = 0; i < text.length(); ++i) {
        if (!text.at(i).isLetterOrNumber())
            setFormat(i, 1, Qt::green);
    }
}
```

La fonction **QSyntaxHighlighter::setFormat()** existe en trois versions, avec les signatures suivantes :

```
void setFormat(int start, int count, const QTextCharFormat &format);
void setFormat(int start, int count, const QColor &color);
void setFormat(int start, int count, const QFont &font)
```

Le paramètre start indique l'index de départ dans la chaîne text et count le nombre de caractères. Le troisième paramètre peut être un objet **QTextCharFormat**, un objet **QColor** ou un objet **QFont**.

IV - Garder les traces des informations d'état

Dans les exemples précédents, la coloration de chaque ligne pouvait être faite indépendamment des autres. Pour la plupart des langages, cette hypothèse ne peut tenir. Voici l'exemple d'une coloration syntaxique C++. Si l'utilisateur

Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

ouvre un style de commentaire C à la ligne 10 et le ferme à la ligne 15, les lignes présentes entre les deux devront aussi être colorées différemment même si elles ne font pas partie du commentaire.

QSyntaxHighlighter rend cela possible au travers d'une machine à état. Quand la coloration d'une ligne est terminée, un état peut être associé à cette ligne (par exemple, « dans un commentaire de style C »), qui peut être récupéré quand la ligne suivante démarre. L'état est enregistré dans une variable de type int.

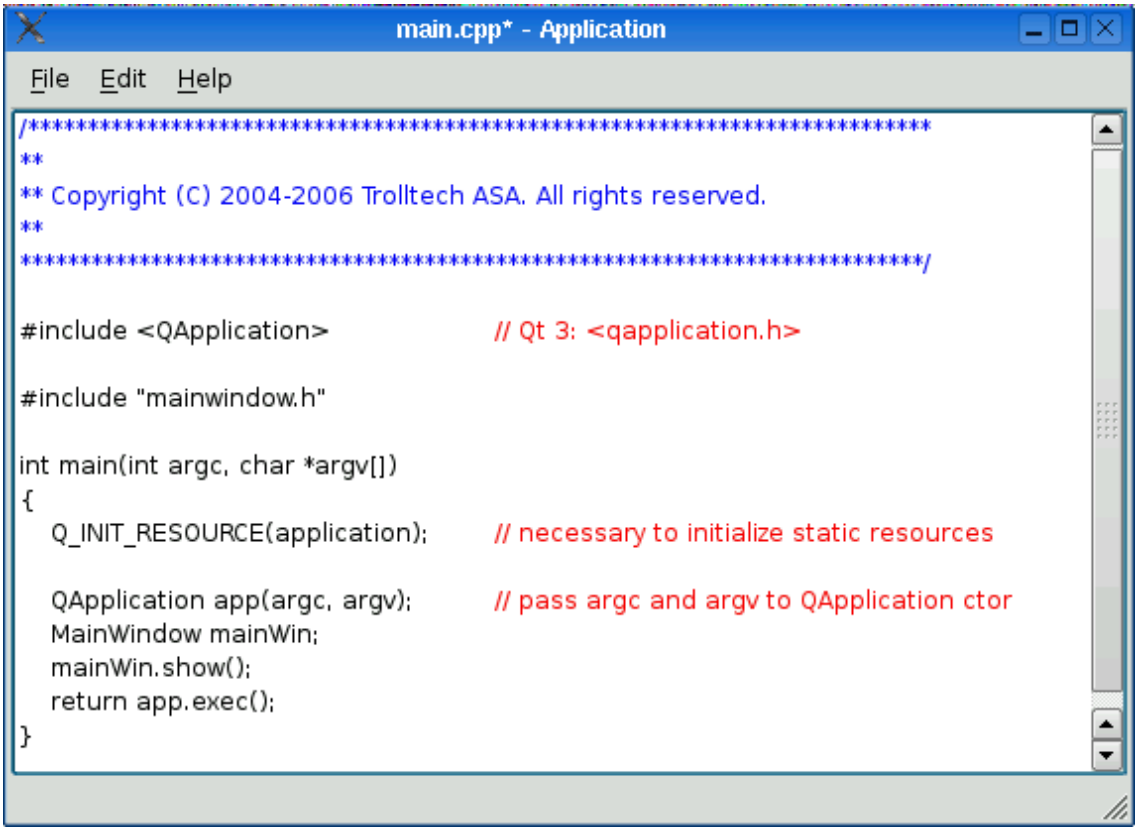
Le code suivant montre comment gérer les commentaires de C et C++ dans une coloration syntaxique de code C++. Le système de coloration dispose de deux états : NormalState et InsideCStyleComment. L'état NormalState est défini à -1, car l'état de **QSyntaxHighlighter** au début du document est toujours à -1.

```
void CppHighlighter::highlightBlock(const QString &text)
{
    enum { NormalState = -1, InsideCStyleComment };

    int state = previousBlockState();
    int start = 0;

    for (int i = 0; i < text.length(); ++i) {
        if (state == InsideCStyleComment) {
            if (text.mid(i, 2) == "*/") {
                state = NormalState;
                setFormat(start, i - start + 2, Qt::blue);
            }
        } else {
            if (text.mid(i, 2) == "//") {
                setFormat(i, text.length() - i, Qt::red);
                break;
            } else if (text.mid(i, 2) == "/*") {
                start = i;
                state = InsideCStyleComment;
            }
        }
    }
    if (state == InsideCStyleComment)
        setFormat(start, text.length() - start, Qt::blue);

    setCurrentBlockState(state);
}
```

Au début de la fonction, l'état de la ligne précédente est récupéré en utilisant **QSyntaxHighlighter::previousBlockState()** et est enregistré dans la variable locale state. Ensuite, les caractères suivants sont itérés et l'état est mis à jour si nécessaire si les chaînes /* et // sont rencontrées.

Les commentaires commençant par // sont aussi colorés. Comme ces commentaires ne peuvent pas être multilignes, il n'y a pas besoin de gérer un état pour ceux-ci. À la fin de la fonction, la fonction setCurrentBlockState() est appelée avec le nouvel état afin qu'il soit disponible pour les lignes suivantes.

Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.



Exemple de coloration de commentaires en C++

En plus de la sauvegarde de l'état, on peut également appeler la fonction setFormat() pour colorer les commentaires C en bleu et les commentaires C++ en rouge.

L'exemple de coloration syntaxique fourni avec Qt 4.2 expose également la plupart de ces principes comme cet exemple, mais utilise des expressions régulières pour localiser les différents symboles dans le code source C++. Cette approche d'utilisation des états par **QSyntaxHighlighter** permet d'avoir un certain degré de flexibilité dans le système de coloration qui est implémenté.

V - Un surligneur de code HTML basique

Voici maintenant un exemple complet et plus complexe d'un code pour colorer des entités HTML (par exemple, « — »), des balises (par exemple, « <p> ») et des commentaires (par exemple, « < !-- commentaire --> »).

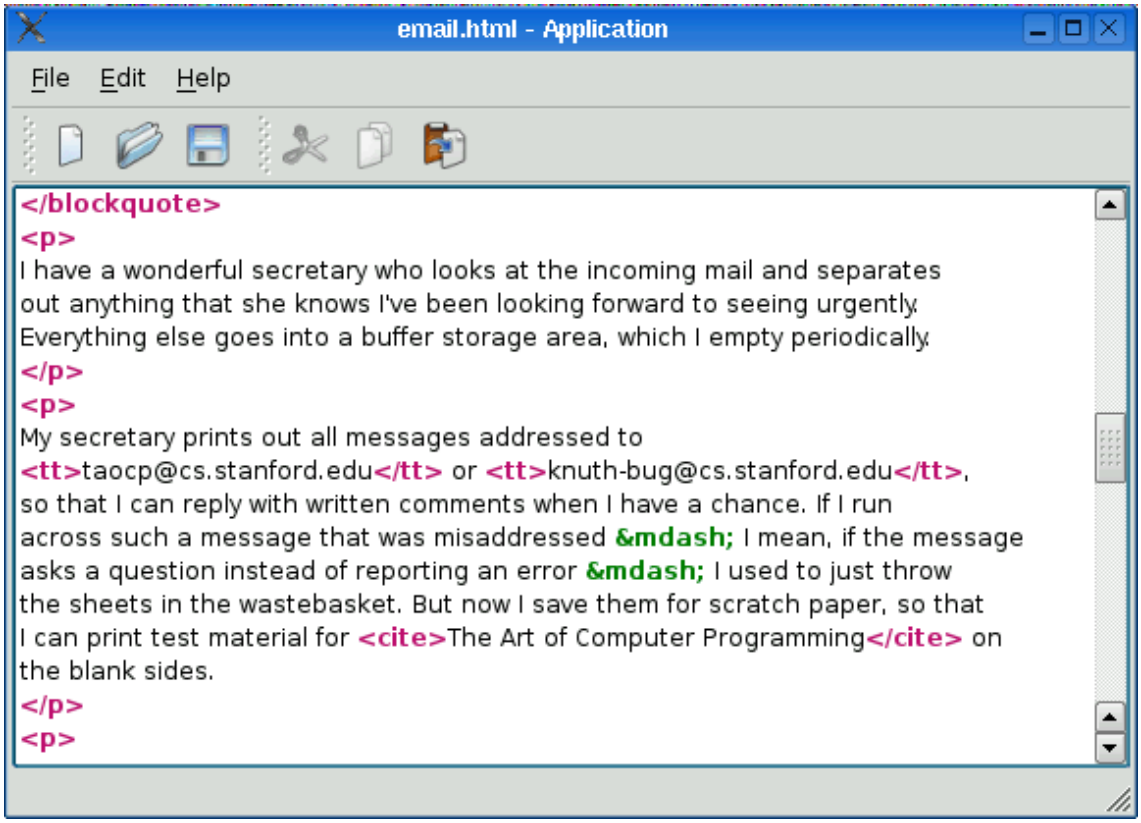
En plus de réimplémenter highlightBlock(), comme dans le premier exemple, des fonctions vont être fournies pour laisser les utilisateurs de la classe spécifier quelles couleurs utiliser pour les divers éléments HTML.

```
class HtmlHighlighter : public QSyntaxHighlighter
{
    Q_OBJECT

public:
    enum Construct {
        Entity,
        Tag,
        Comment,
        LastConstruct = Comment
    };

    HtmlHighlighter(QTextDocument *document);

    void setFormatFor(Construct construct,
```


Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

```
                        const QTextCharFormat &format);
    QTextCharFormat formatFor(Construct construct) const
        { return m_formats[construct]; }

protected:
    enum State {
        NormalState = -1,
        InComment,
        InTag
    };

    void highlightBlock(const QString &text);

private:
    QTextCharFormat m_formats[LastConstruct + 1];
};
```

Les fonctions setFormatFor() et formatFor() permettent de changer le formatage utilisé pour les éléments HTML supportés (entités, balises et commentaires). Le type énuméré State spécifie les trois états que l'analyseur HTML peut utiliser après avoir analysé une ligne. L'état NormalState est fixé à -1, l'état par défaut dans **QSyntaxHighlighter**.



Exemple de coloration HTML

Voici l'implémentation du constructeur :

```
HtmlHighlighter::HtmlHighlighter(QTextDocument *document)
    : QSyntaxHighlighter(document)
{
    QTextCharFormat entityFormat;
    entityFormat.setForeground(QColor(0, 128, 0));
    entityFormat.setFontWeight(QFont::Bold);
    setFormatFor(Entity, entityFormat);

    QTextCharFormat tagFormat;
    tagFormat.setForeground(QColor(192, 16, 112));
    tagFormat.setFontWeight(QFont::Bold);
```


Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

```
    setFormatFor(Tag, tagFormat);

    QTextCharFormat commentFormat;
    commentFormat.setForeground(QColor(128, 10, 74));
    commentFormat.setFontItalic(true);
    setFormatFor(Comment, commentFormat);
}
```

Dans le constructeur, les différents formatages sont définis pour les entités, balises et commentaires HTML. Un format est représenté par un objet **QTextCharFormat**. Les propriétés sont spécifiées avec le texte surligné - la couleur de fond, le style de la police (gras ou non), le style de soulignement (simple, ondulé), etc. - et ceci s'applique aux attributs préexistants dans la classe **QTextDocument**.

```
void HtmlHighlighter::setFormatFor(Construct construct,
                           const QTextCharFormat &format)
{
    m_formats[construct] = format;
    rehighlight();
}
```

La fonction setFormatFor() adapte l'instance de la classe **QTextCharFormat** à un élément HTML donné. La fonction **QSyntaxHighlighter::rehighlight()** est appelée pour appliquer immédiatement les changements dans tout le document.

La seule fonction qui reste à voir est highlightBlock() qui est la ré-implémentation de **QSyntaxHighlighter**. C'est une fonction importante, elle va donc être étudiée pas à pas.

```
void HtmlHighlighter::highlightBlock(const QString &text)
{
    int state = previousBlockState();
    int len = text.length();
    int start = 0;
    int pos = 0;
```

Comme dans l'exemple de coloration C++, la fonction previousBlockState() est appelée pour récupérer l'état de la ligne précédente. Dans la boucle, le nouvel état est basculé et les caractères sont interprétés différemment et indépendamment selon qu'ils sont dans une balise ou dans un commentaire.

```
    while (pos < len) {
        switch (state) {
        case NormalState:
        default:
            while (pos < len) {
                QChar ch = text.at(pos);
                if (ch == '<') {
                    if (text.mid(pos, 4) == "<!--") {
                        state = InComment;
                    } else {
                        state = InTag;
                    }
                    break;
                } else if (ch == '&') {
                    start = pos;
                    while (pos < len
                           && text.at(pos++) != ';')
                        ;
                    setFormat(start, pos - start,
                              m_formats[Entity]);
                } else {
                    ++pos;
                }
            }
            break;
```

Pour l'état NormalState, on avance jusqu'à un début de balise (<) ou une esperluette (&).

Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

Si un début de balise (<) est trouvé, l'état Intag ou InComment est fixé. Si une esperluette (&) est trouvée, on a affaire à une entité HTML et elle est formatée si besoin (un état InEntity peut être utilisé dans un « switch case », mais cela n'est pas nécessaire, car les entités, par rapport aux balises et aux commentaires, ne peuvent être découpées sur plusieurs lignes).

```
            case InComment:
                start = pos;
                while (pos < len) {
                    if (text.mid(pos, 3) == "-->") {
                        pos += 3;
                        state = NormalState;
                        break;
                    } else {
                        ++pos;
                    }
                }
                setFormat(start, pos - start,
                          m_formats[Comment]);
                break;
```

Si le caractère se trouve dans un commentaire, on continue jusqu'à trouver la balise fermante (-->). Si cette balise est trouvée, l'objet retrouve l'état NormalState. Dans tous les cas, les commentaires sont surlignés en utilisant la fonction QSyntaxHighlighter::setFormat().

```
    case InTag:
                QChar quote = QChar::Null;
                start = pos;
                while (pos < len) {
                    QChar ch = text.at(pos);
                    if (quote.isNull()) {
                        if (ch == '\" || ch == '"') {
                            quote = ch;
                        } else if (ch == '>') {
                            ++pos;
                            state = NormalState;
                            break;
                        }
                    } else if (ch == quote) {
                        quote = QChar::Null;
                    }
                    ++pos;
                }
                setFormat(start, pos - start, m_formats[Tag]);
            }
        }
```

Si le caractère se trouve à l'intérieur d'une balise, on détecte alors le « > » suivant, en sautant chaque attribut et valeur entre guillemets (par exemple, <img alt= »>>> »>). Si le caractère « > » est trouvé, on fixe alors l'état à NormalState. Dans tous les cas, on colore la balise.

```
    setCurrentBlockState(state);
}
```

À la fin de la fonction, l'état courant est enregistré pour que la prochaine ligne débute avec le bon état.

Pour utiliser le système de coloration, il faut simplement l'instancier avec **QTextDocument** comme parent. Par exemple :

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QTextEdit editor;
    HtmlSyntaxHighlighter highlighter(editor.document());
    editor.show();
    return app.exec();
```


Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.

```
}
```

Pour améliorer encore plus la coloration syntaxique HTML, différents surlignages peuvent être utilisés pour les délimiteurs de balises, les noms, attributs et valeur d'attributs.

VI - Résumé

QSyntaxHighlighter permet facilement d'écrire des systèmes de coloration syntaxique interactifs. Quand l'utilisateur édite son texte, celui-ci va recolorer les parties du texte qui doivent être mises à jour. Par exemple, si l'utilisateur modifie la ligne 3 d'un document comportant 2 000 lignes, **QSyntaxHighlighter** commencera à recolorer la ligne 3 et continuera seulement si l'état associé à la ligne 3 a changé comme un résultat et s'arrête à la première ligne dont l'ancien état est identique au nouveau.

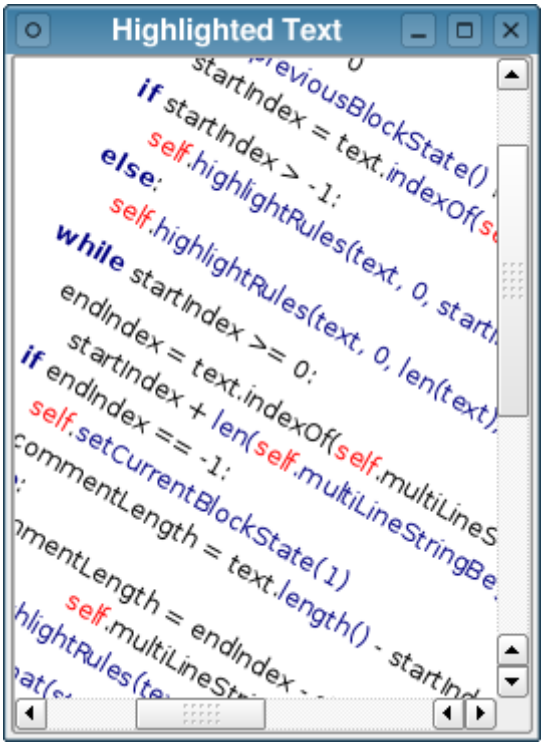


Figure 3 - exemple de coloration syntaxique en Python

Dans Qt4, **QSyntaxHighlighter** n'est pas restreint à **QTextEdit**. Il peut être utilisé avec des éditeurs basés sur **QTextDocument**., incluant le nouveau **QGraphicsTextItem** pour **QGraphicsView**, et permettant de fournir la coloration syntaxique de texte et la facilité d'édition pour les textes transformés.

La nouvelle classe **QSyntaxHighlighter** est aussi extrêmement flexible quand elle est utilisée pour manipuler les formats de caractères, merci à la puissance de la classe **QTextCharFormat**.

VII - Remerciements

Au nom de toute l'équipe Qt, j'aimerais adresser le plus grand remerciement à Nokia pour nous avoir autorisés à traduire cet article !

Je tiens à remercier **Thibaut Cuvelier** pour ses conseils et **Jacques Thery** pour sa relecture.

Copyright ® 2007 Nokia. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.