Utiliser QScintilla

Par Eclyps

Date de publication : 10 novembre 2008

Dernière mise à jour : 27 décembre 2019

Dans ce tutoriel nous allons apprendre à colorier syntaxiquement les textes dans Qt.
Vos questions et remarques sur le forum **Commentez**

I - Introduction....3
II - Installer QScintilla....3
III - Utiliser QScintilla....4
IV - Colorions nos textes....5
V - Lignes numérotées....7
VI - QScintilla dans Qt Designer....8

I - Introduction


```
1  /* Test*/
2
3  #include <QApplication>
4  #include <Qsci/qsciscintilla.h>
5
6  int main(int argc, char* argv[])
7  {
8          QApplication app(argc, argv);
9          QsciScintilla *scintilla = new QsciScintilla;
10         scintilla->show();
11         return app.exec();
12 }
13
14
```

QScintilla de Riverbank est une bibliothèque complémentaire de Qt permettant la coloration syntaxique du texte, mais aussi beaucoup d'autres choses comme :

- la marge ;
- la complétion de code ;
- les indicateurs d'erreur dans la marge ;
- les indicateurs de chaine non fermée ;

Et beaucoup d'autres choses !

QScintilla est sous la même licence que Qt (GNU GPL v2, v3 et sous licence Commercial).

II - Installer QScintilla

Pour ceux qui sont sous Linux, vous pouvez télécharger le paquet : libqscintilla2-dev.
Pour les autres téléchargez-le ici :
http://www.riverbankcomputing.com/software/qscintilla/download, puis décompressez l'archive.

S'il y a un problème, j'ai hébergé les fichiers compressés de QScintilla sur mon serveur FTP :

- Windows (.zip) : **QScintilla-gpl-2.3.2.zip**
- Linux, macOS (.tar.gz) : **QScintilla-gpl-2.3.2.tar.gz**

QScintilla peut s'installer en bibliothèque dynamique ou en bibliothèque statique.

Installation en bibliothèque dynamique :

- Ouvrez la console Qt et déplacez-vous dans le dossier Qt 4 de QScintilla-gpl-2.3.2, puis lancez :

- *qmake* ;
- *make* ;
- *make install* ;
- ou *qmake & make & make install* (mais seulement sur Windows).

Les bibliothèques seront dans le dossier debug (ou release).

Installation en bibliothèque statique :

- ouvrez le dossier *Qscintilla-gpl-2.3.2* ;
- puis ouvrez le dossier *Qt4* ;
- ouvrez le fichier *qscintilla.pro :*
    - faites une recherche de *CONFIG += qt warn_off release dll thread,*
    - changez *dll* par *staticlib,*
    - faites une autre recherche de *DEFINES = QSCINTILLA_MAKE_DLL QT SCI_LEXER,*
    - supprimez *QSCINTILLA_MAKE_DLL,*
    - ou alors, téléchargez et utilisez le fichier que j'ai préconfiguré : **qscintilla.pro** ;
- ouvrez une console Qt ;
- déplacez-vous dans le dossier *Qt4* de *Qscintilla-gpl-2.3.2* ;
- faites :
    - *qmake,*
    - *make,*
    - *make install* ;
    - ou alors : *qmake & make & make install* (mais seulement sur Windows).

Si vous avez une erreur semblable à celle-ci :

```
qsciscintilla.cpp: In member function `bool QsciScintilla::isModified() const':
qsciscintilla.cpp:1674: error: 'const class QsciDocument' has no member named 'isModified'
qsciscintilla.cpp: In member function `void QsciScintilla::handleSavePointReached()':
qsciscintilla.cpp:1712: error: 'class QsciDocument' has no member named 'setModified'
qsciscintilla.cpp: In member function `void QsciScintilla::handleSavePointLeft()':
qsciscintilla.cpp:1720: error: 'class QsciDocument' has no member named 'setModified'
qsciscintilla.cpp: In member function `bool QsciScintilla::write(QIODevice*) const':
qsciscintilla.cpp:3409: error: `SCI_GETCHARACTERPOINTER' undeclared (first use this function)
qsciscintilla.cpp:3409: error: (Each undeclared identifier is reported only once for each function it appears in.)
mingw32-make[1]: *** [release/qsciscintilla.o] Error 1mingw32-make[1]: Leaving directory `C:/cygwin/QScintilla-gpl-2.3.1/Qt4
mingw32-make: *** [release] Error 2
```

Erreur

Réinstallez Qt (tout compris même QScintilla si vous l'aviez auparavant) puis réinstallez-le. Et quand vous faites make install, il devrait y avoir des erreurs : ignorez-les, c'est tout à fait normal, ne vous en faites pas.

III - Utiliser QScintilla

La documentation de QScintilla se trouve ici : **http://www.riverbankcomputing.com/static/Docs/QScintilla2/annotated.html**
Avec les bibliothèques dynamiques, ajoutez-les dans le dossier debug (ou release).
Il faut aussi les linker : dans le .pro ajouter au-dessus de *# Input* :

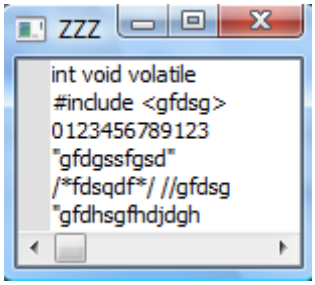
- *LIBS += -lqscintilla2*

On utilise la classe *QsciScintilla(QWidget *parent=0)* contenue dans *#include <Qsci/qsciscintilla.h>* pour créer une zone de texte.
Ce qui nous donne :

```
#include <QApplication>
#include <Qsci/qsciscintilla.h>
```

```
int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QsciScintilla *scintilla = new QsciScintilla;
    scintilla->show();
    return app.exec();
}
```

Si l'on compile, on devrait se retrouver avec :



IV - Colorions nos textes

Il y a énormément de langages que peut colorier syntaxique QScintilla, les voici :

1 Bash ;
2 Batch ;
3 CMake ;
4 CPP ;
5 CSharp ;
6 CSS ;
7 D ;
8 Diff ;
9 Fortran ;
10 Fortran77 ;
11 HTML ;
12 IDL ;
13 Java ;
14 JavaScript ;
15 Lua ;
16 Makefile ;
17 Pascal ;
18 Perl ;
19 PostScript ;
20 POV ;
21 Properties ;
22 Python ;
23 Ruby ;
24 SQL ;
25 TCL ;
26 TeX ;
27 VHDL ;
28 XML ;
29 YAML.

Dans QScintilla l'objet pour colorier la syntaxe d'un QsciScintilla s'appelle un lexer.
Le nom de la classe du lexer de C++ est : QsciLexerCPP.
Celui de Java c'est : QsciLexerJava.

Les fichiers à inclure sont donc pour le lexer de C++ : *#include<Qsci/qscilexercpp.h>*

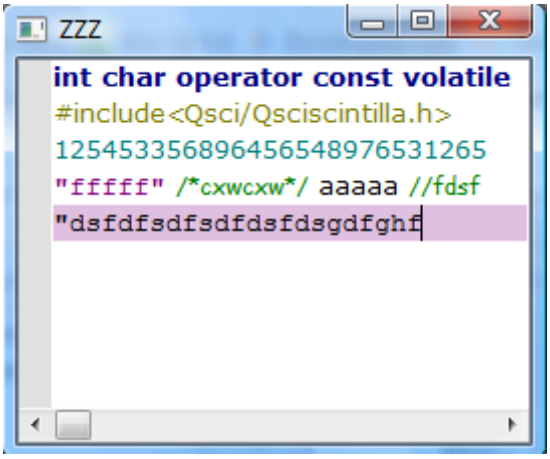
Pour attribuer un lexer à un QsciScintilla, il faut utiliser la méthode : *virtual void QsciScintilla::setLexer (QsciLexer *lexer=0)*

Voici un exemple de code que l'on peut obtenir :

```
#include <QApplication>
#include <Qsci/qsciscintilla.h>
#include <Qsci/qscilexercpp.h>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QsciScintilla *scintilla = new QsciScintilla;
    QsciLexerCPP *lexerCPP = new QsciLexerCPP;
    scintilla->setLexer(lexerCPP);
    scintilla->show();
return app.exec();
}
```

Une petite capture d'écran :) :



Il est aussi possible de modifier les couleurs des lexers.
Sur la documentation de **QsciLexerCPP**, il y a l'énumération :

```
Default = 0, Comment = 1, CommentLine = 2,
CommentDoc = 3, Number = 4, Keyword = 5,
DoubleQuotedString = 6, SingleQuotedString = 7, UUID = 8,
PreProcessor = 9, Operator = 10, Identifier = 11,
UnclosedString = 12, VerbatimString = 13, Regex = 14,
CommentLineDoc = 15, KeywordSet2 = 16, CommentDocKeyword = 17,
CommentDocKeywordError = 18, GlobalClass = 19
```

Ce sont les mots-clés que l'on peut modifier :

- Default = azertyuiopqsdfghjklmwxcvbn ;
- Comment = /* azertyuiopqsdfghjklmwxcvbn */
- CommentLine = // azertyuiopqsdfghjklmwxcvbn
- Number = 0123456789
- Keyword = int const mutable void operator
- DoubleQuotedString = "blablabla"
- SingleQuotedString = 'blablabla'
- PreProcessor = #include #define #ifndef
- Operator = + - * / = << >>

- UnclosedString = Quand une chaine n'est pas fermée "blablabla

Pour changer la couleur, il faut utiliser la méthode : *virtual void setColor (const QColor &c, int style=-1)*

Exemple : pour avoir les mots-clés en violet :

```
QsciLexerCPP lexer;
lexer.setColor(Qt::magenta, QsciLexerCPP::Keyword);
```

Si vous ne trouvez pas d'énumération pour la couleur dans la classe d'un lexer, regardez dans sa classe mère.

V - Lignes numérotées

Dans QScintilla, il est possible d'ajouter une marge qui contient les numéros de ligne des zones de texte.

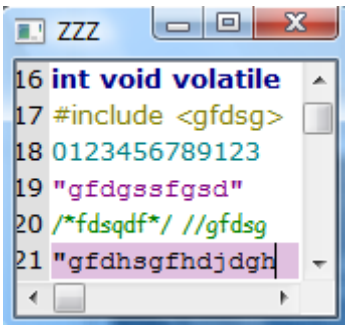
Pour afficher la marge, il faut utiliser : *virtual void setMarginLineNumbers(int margin, bool lnrs)*

(Il peut y avoir plusieurs marges.)
Voici un exemple de code possible :

```
#include <QApplication>
#include <Qsci/qsciscintilla.h>
#include <Qsci/qscilexercpp.h>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QsciScintilla *scintilla = new QsciScintilla;
    QsciLexerCPP *lexerCPP = new QsciLexerCPP;
    scintilla->setLexer(lexerCPP);
    scintilla->setMarginLineNumbers(1,true);
    scintilla->show();
return app.exec();
}
```

Résultat :



On peut agrandir la marge avec : *virtual void setMarginWidth (int margin, int width)*

width étant la largeur de la marge en pixel et margin le numéro de la marge.

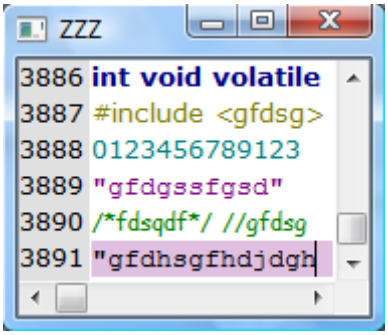
Ou bien : *virtual void setMarginWidth (int margin, const QString &s)*

s étant la largeur de la marge souhaitée en caractères + 1. Par exemple, si l'on veut faire une marge de 4 caractères il faut faire 4 + 1 = 5 caractères :
setMarginWidth(1, "-----") (4 caractères + 1)
Testons cela :

```
#include <QApplication>
#include <Qsci/qsciscintilla.h>
#include <Qsci/qscilexercpp.h>

int main(int argc, char* argv[])
{
    QApplication app(argc, argv);
    QsciScintilla *scintilla = new QsciScintilla;
    QsciLexerCPP *lexerCPP = new QsciLexerCPP;
    scintilla->setLexer(lexerCPP);
    scintilla->setMarginLineNumbers (1, true);
    scintilla->setMarginWidth(1, "-----");
    // scintilla->setMarginWidth(1, 50);
    scintilla->show();
return app.exec();
}
```

Résultat :



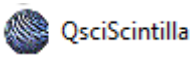
Il existe encore plein de méthodes utiles/inutiles dans QScintilla. Si vous voulez les connaître, allez voir la **documentation**.

VI - QScintilla dans Qt Designer

Une dernière chose : il est possible d'inclure QScintilla dans Qt Designer :

- depuis la console de Qt, déplacez-vous dans le dossier *designer-Qt4* de *QScintilla-gpl-2.3.2*
- puis faites :
    - qmake,
    - make,
    - make install ;
    - ou bien : qmake & make & make install (mais seulement sur Windows).

Ouvrez Qt Designer, cherchez un peu dans les widgets, et magie ! dans Input Widget il y a QsciScintilla :



Pour ajouter des lexers, il est nécessaire de les coder (ce serait trop beau sinon).

Voilà cet article est fini.
J'espère qu'il vous a plu.