

Générer du XML

Qt by Nokia

par Jasmin Blanchette traducteur : Thibaut Cuvelier Qt Quarterly

Date de publication : 18/06/2009


Dernière mise à jour : 06/07/2010

Qt fournit des classes DOM et SAX pour l'analyse du XML, mais n'a pas toujours proposé une classe pour sa génération. Dans cet article, nous allons développer une classe XmlWriter, basée sur **QTextStream**. Nous allons exemplifier son usage avec un générateur de fichiers .ui.

Cet article est une traduction autorisée de **Generating XML**, par Jasmin Blanchette.

I - L'article original.....	3
II - Définition de la classe XmlWriter.....	3
III - Générer un fichier XML.....	4
IV - Implémentation de la classe XmlWriter.....	5
V - Suggestions d'améliorations.....	7
VI - Divers.....	7

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 **Generating XML** de Jasmin Blanchette paru dans la Qt Quarterly Issue 5.

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

II - Définition de la classe XmlWriter

La classe XmlWriter fournit une API basique pour la génération de fichiers XML. Pour commencer, voici la définition de cette classe.

```
class XmlWriter
{
public:
    XmlWriter( QIODevice *device,
               const QTextCodec *codec = 0 );
    XmlWriter();

    void writeRaw( const QString& xml );
    void writeString( const QString& string );
    void writeOpenTag( const QString& name,
                      const AttrMap& attrs = AttrMap() );
    void writeCloseTag( const QString& name );
    void writeAtomTag( const QString& name,
                      const AttrMap& attrs = AttrMap() );
    void writeTaggedString( const QString& name,
                           const QString& string,
                           const AttrMap& attrs = AttrMap() );

    void newLine();
    void setIndentSize( int size ) { indentSize = size; }
    void setAutoNewLine( bool on ) { autoNewLine = on; }

private:
    ...

    QTextStream out;
    QString indentStr;
    int indentSize;
    bool autoNewLine;
    bool atBeginningOfLine;
};
```

Le paramètre AttrMap, de writeOpenTag(), writeAtomTag() et writeTaggedString(), stocke des paires nom = valeur. Cette classe hérite ses propriétés de **QMap**.

```
class AttrMap : public QMap<QString, QString>
{
public:
    AttrMap() { }
    AttrMap( const QString& name, const QString& value )
    {
        insert( name, value );
    }
};
```

Par exemple, cette ligne peut être générée par ce code.

```
<animal age="25">Elephant</animal>
```

```
XmlWriter xw( device );  
xw.writeTaggedString( "animal", "Elephant", AttrMap("age", "25") );
```

III - Générer un fichier XML

Avant d'implémenter cette classe, nous allons voir comment l'utiliser pour générer un fichier XML.

```
<!DOCTYPE UI><UI version="3.1">  
<class>Form1</class>  
<widget class="QDialog">  
  <property name="name">  
    <cstring>Form1</cstring>  
  </property>  
  <property name="caption">  
    <string>Form1</string>  
  </property>  
  <vbox>  
    <widget class="QLabel">  
      <property name="name">  
        <cstring>label</cstring>  
      </property>  
      <property name="text">  
        <string>Rock &amp;&amp; Roll</string>  
      </property>  
    </widget>  
  </vbox>  
</widget>  
<layoutdefaults margin="11" spacing="6"/>  
</UI>
```

Le code qui génère cet exemple suit. Sa longueur peut paraître exagérée au vu de la sortie produite, de taille inférieure à celle du programme.

Cependant, dans un contexte plus réaliste, où la sortie dépend de l'entrée et de l'état du programme, cette classe va permettre de sauver un temps précieux, et empêchera toutes les bogues ennuyantes, parce qu'elle échappe les caractères spéciaux en XML automatiquement.

```
void writeProperty( XmlWriter& xw, const QString& name,  
                  const QString& type, const QString& value )  
{  
    xw.writeOpenTag( "property", AttrMap("name", name) );  
    xw.writeTaggedString( type, value );  
    xw.writeCloseTag( "property" );  
}  
  
int main()  
{  
    QFile file;  
    file.open( IO_WriteOnly, stdout );  
    XmlWriter xw( & file );  
    xw.setAutoNewLine( true );  
    xw.writeRaw( "<!DOCTYPE UI><UI version=\"3.1\">" );  
    xw.newLine();  
    xw.writeTaggedString( "class", "Form1" );  
    xw.writeOpenTag( "widget", AttrMap("class", "QDialog") );  
    writeProperty( xw, "name", "cstring", "Form1" );  
    writeProperty( xw, "caption", "string", "Form1" );  
    xw.writeOpenTag( "vbox" );  
    xw.writeOpenTag( "widget", AttrMap("class", "QLabel") );  
    writeProperty( xw, "name", "cstring", "label" );  
    writeProperty( xw, "text", "string", "Rock && Roll" );  
    xw.writeCloseTag( "widget" );  
}
```

```
xw.writeCloseTag( "vbox" );
xw.writeCloseTag( "widget" );
AttrMap attrs;
attrs.insert( "spacing", "6" );
attrs.insert( "margin", "11" );
xw.writeAtomTag( "layoutdefaults", attrs );
xw.writeRaw( "</UI>" );
return 0;
}
```

Avec ce code, nous obtenons un code déjà bien formaté (indentation et retours à la ligne) sans effort, et nous ne devons pas nous occuper des caractères spéciaux, qui doivent être écrits comme des entités XML (comme &, ou bien des caractères Unicode. Soit XmlWriter s'en occupe, soit **QTextStream**.

IV - Implémentation de la classe XmlWriter

Maintenant, jetons un coup d'oeil à l'implémentation de XmlWriter.

Le constructeur initialise le flux de texte out avec un périphérique d'E/S et initialise d'autres variables membres privées. Si un codec est fourni, le flux l'utilise, et un en-tête est ajouté pour le préciser. Par défaut, un texte est encodé en Unicode UTF-8/UTF-16.

```
XmlWriter::XmlWriter( QIODevice *device,
                     const QTextCodec *codec )
: indentSize( 4 ), autoNewLine( false ),
  atBeginningOfLine( true )
{
    out.setDevice( device );
    if ( codec == 0 )
    {
        out.setEncoding( QTextStream::UnicodeUTF8 );
    }
    else
    {
        out.setCodec( codec );
        out << "<?xml version=\"1.0\" encoding=\""
              << protect( codec->mimeTypeName() ) << "\"?>\n";
    }
}
```

Le destructeur ajoute une ligne vide en mode autoNewLine. En effet, les systèmes UNIX préconisent l'utilisation d'une ligne vide (\n) en fin de fichier.

```
XmlWriter::~XmlWriter()
{
    if ( autoNewLine && !atBeginningOfLine )
        out << endl;
}
```

Même si elle est privée, la fonction protect() est de loin la plus utile. Elle remplace les caractères par des identités.

```
QString XmlWriter::protect( const QString& string )
{
    QString s = string;
    s.replace( "&", "&amp;" );
    s.replace( ">", "&gt;" );
    s.replace( "<", "&lt;" );
    s.replace( "\"", "&quot;" );
    s.replace( "'", "&apos;" );
    return s;
}
```

La méthode opening() privée construit une balise ouvrante avec le nom tag précisé et les attributs attrs.

```
QString XmlWriter::opening( const QString& tag,
                           const AttrMap& attrs )
{
    QString s = "<" + tag;
    AttrMap::ConstIterator a = attrs.begin();
    while ( a != attrs.end() )
    {
        s += " " + a.key() + "=\"" + protect( *a ) + "\"";
        ++a;
    }
    s += ">";
    return s;
}
```

writePendingIndent() indente une balise.

```
void XmlWriter::writePendingIndent()
{
    if ( atBeginningOfLine )
    {
        out << indentStr;
        atBeginningOfLine = false;
    }
}
```

Maintenant, nous allons passer aux fonctions publiques.

newLine() peut être utilisée pour forcer une nouvelle ligne.

```
void XmlWriter::newLine()
{
    out << endl;
    atBeginningOfLine = true;
}
```

writeRaw écrit du XML pur.

```
void XmlWriter::writeRaw( const QString& xml )
{
    out << xml;
    atBeginningOfLine = false;
}
```

writeString() écrit une chaîne en remplaçant les caractères spéciaux par leurs entités XML.

```
void XmlWriter::writeString( const QString& string )
{
    out << protect( string );
    atBeginningOfLine = false;
}
```

writeOpenTag() ouvre une balise avec des attributs optionnels. Par exemple, `<item id="23">`

```
void XmlWriter::writeOpenTag( const QString& name,
                             const AttrMap& attrs )
{
    writePendingIndent();
    out << opening( name, attrs );
    indentStr += QString().fill( ' ', indentSize );
    if ( autoNewLine )
        newLine();
}
```

writeCloseTag() ferme un tag. Par exemple, `</item>`.

```
void XmlWriter::writeCloseTag( const QString& name )
{
    indentStr = indentStr.mid( indentSize );
    writePendingIndent();
    out << opening( "/" + name );
    if ( autoNewLine )
        newLine();
}
```

`writeAtomTag()` crée une balise, ouverte puis fermée. Par exemple, `<item id="23" />`

```
void XmlWriter::writeAtomTag( const QString& name,
                             const AttrMap& attrs )
{
    writePendingIndent();
    QString atom = opening( name, attrs );
    atom.insert( atom.length() - 1, "/" );
    out << atom;
    if ( autoNewLine )
        newLine();
}
```

`writeTaggedString()` simplifie une partie des opérations : elle ouvre une balise, y insère un texte et referme la balise.

```
void XmlWriter::writeTaggedString( const QString& name,
                                   const QString& string,
                                   const AttrMap& attrs )
{
    writePendingIndent();
    out << opening( name, attrs );
    writeString( string );
    out << opening( "/" + name );
    if ( autoNewLine )
        newLine();
}
```

V - Suggestions d'améliorations

Cette classe `XmlWriter` est déjà fort utile, mais beaucoup d'améliorations pourraient être effectuées. Voici quelques suggestions.

- Émettre un avertissement si les balises ne sont pas équilibrées ;
- Supporter une plus grande partie du XML sans devoir écrire en dur ;
- Utiliser **`QVariant`** au lieu de **`QString`** pour le type de l'attribut dans `AttrMap`

Vous pouvez télécharger [Sources les sources](#) de cet article. Ces sources incluent la classe `XmlWriter` avec les petites applications de test.

VI - Divers

Ceci a été écrit pour Qt3, mais il reste intéressant pour comprendre le mode de fonctionnement de **`QXmlStreamWriter`**

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