

Écrire des fichiers ODF avec Qt

Qt by Nokia

par Thomas Zander traducteur : Nob_. Qt Quarterly

Date de publication : 05/08/2009


Dernière mise à jour : 10/07/2010

L'arrivée de Qt 4.5 marque l'apparition de la classe **QTextDocumentWriter**, rendant possible la création de fichiers au format OpenDocument (ODF) à partir de n'importe quel document Qt. Ceci ouvre la voie à la création automatisée de documents et à la distribution dans un format conforme aux standards que l'utilisateur pourra ouvrir dans une large gamme de logiciels de traitement de texte.

Cet article est une traduction autorisée de **Writing ODF Files with Qt**, de Thomas Zander.

I - L'article original.....	3
II - Introduction.....	3
III - Commencer.....	4
IV - Écrire la note.....	4
V - Conclusions.....	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**.

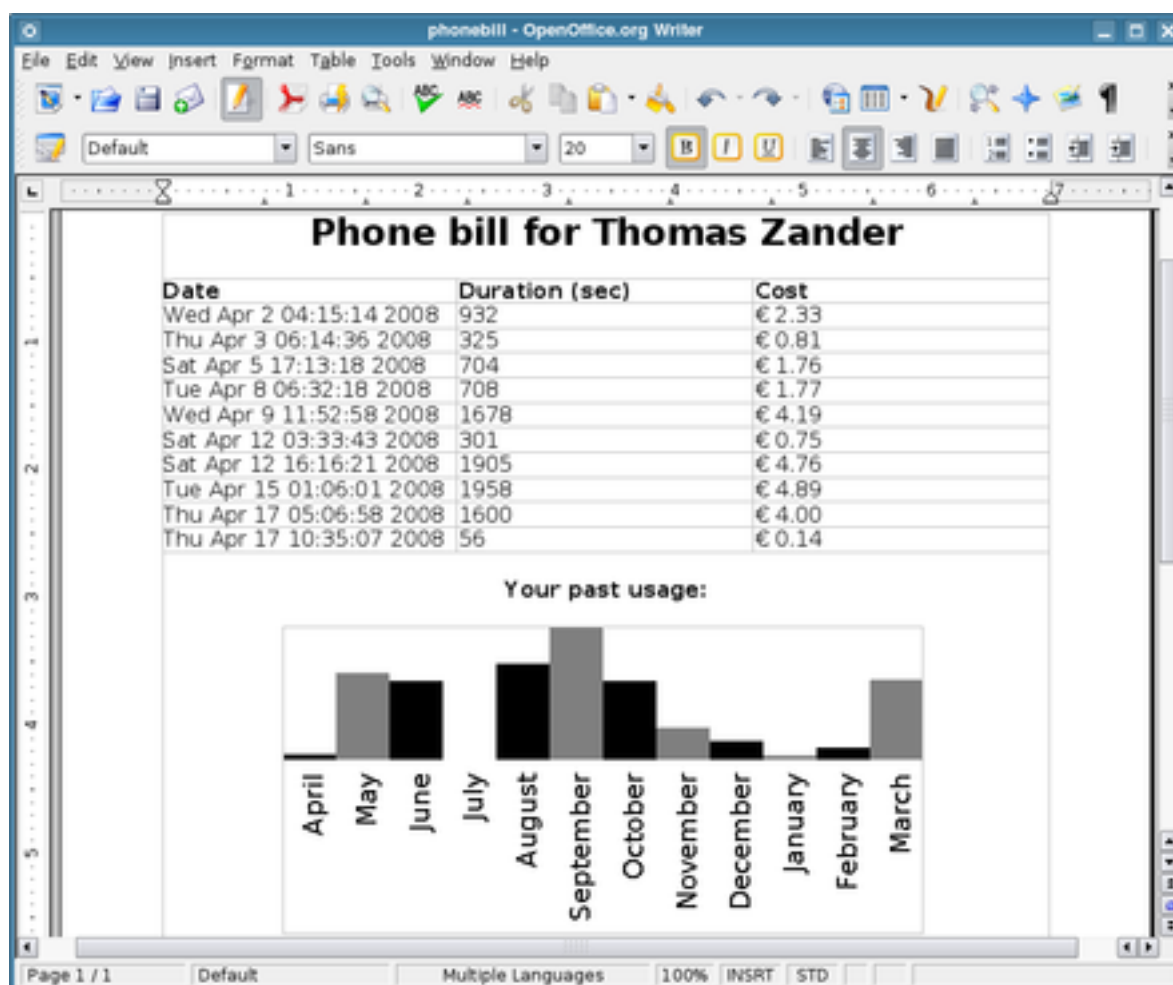
Nokia, Qt, Qt Quarterly et leurs logos sont des marques déposées de *Nokia Corporation* en Finlande et/ou dans les autres pays. Les autres marques déposées sont détenues par leurs propriétaires respectifs.

Cet article est la traduction de l'article **Writing ODF Files with Qt** de Thomas Zander paru dans la  **Qt Quarterly Issue 27**.

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 - Introduction

La génération de rapport est un premier cas d'utilisation de documents automatisés. Par exemple, un enregistrement qui effectue la vérification périodique d'un inventaire et qui, par conséquent, aurait besoin de savoir ce qui est en rayon pour la base de données, est le reflet de la réalité. Pensez à des logiciels qui prendraient les enregistrements de la base de données pour un jour valable de travail et en créerait un document exploitable. Ce document pourra alors être exporté en ODF et envoyé à la personne qui effectue le travail.



Dans cet article nous allons écrire un simple générateur de rapport sous la forme d'un créateur de note téléphonique. Nous créerons une classe, `PhoneBillWriter`, pour représenter la note téléphonique pour un client, et vous pourrez le modifier ou l'utiliser comme point de départ selon vos propres besoins.

III - Commencer

La classe `PhoneBillWriter` se définit ainsi.

```
class PhoneBillWriter
{
public:
    PhoneBillWriter(const QString &client);
    ~PhoneBillWriter();
    struct PhoneCall {
        QDateTime date;
        int duration; // in seconds
        int cost; // in euro-cents
    };
    void addPhoneCall(const PhoneCall &call);
    void addGraph(QList<int> values,
                  const QString &subtext);
    void write(const QString &fileName);

private:
    QTextDocument * const m_document;
    QTextCursor m_cursor;
};
```

Ceci nous permet de créer une instance de `PhoneBillWriter` pour chaque note et de la remplir avec des données, avec l'appel de la méthode `addPhoneCall()`. Une fois toutes nos données ajoutées, nous appelons la méthode `write()` afin de terminer la présente note. C'est un cas classique d'utilisation d'un schéma de construction.

En interne, la classe utilise un **`QTextDocument`** que nous pouvons voir parmi les membres privés. Cette classe **`QTextDocument`** est utilisée à maints endroits dans Qt et ses widgets ; **`QTextEdit`** l'intègre et son accès peut se faire en utilisant la méthode **`QTextEdit::document()`**. **`QTextDocument`** est un document texte ayant la capacité de comporter du texte structuré, formaté (gras et italique) et bien plus encore. Voir la démo *Qt's Text Edit* pour une vue générale.

La classe **`QTextCursor`** est fournie afin de permettre la manipulation du contenu d'un document texte. Ceci afin d'avoir un curseur clignotant à la manière d'un traitement de texte ainsi que la possibilité de déplacer ce curseur et d'insérer du texte. La classe **`QTextCursor`** donne la possibilité de faire tout ceci avec du code.

IV - Écrire la note

L'approche utilisée pour ce créateur de bulletin est d'écrire dans le constructeur un **`QTextDocument`** et d'ajouter en en-tête une présentation agréable de l'information qui sera identique pour chaque note téléphonique. En voici une simple implémentation.

```
PhoneBillWriter::PhoneBillWriter(const QString &client)
    : m_document(new QTextDocument()),
      m_cursor(m_document)
{
    m_cursor.insertText(QObject::tr(
        "Phone bill for %1\n").arg(client));

    QTextTableFormat tableFormat;
    tableFormat.setCellPadding(5);
    tableFormat.setHeaderRowCount(1);
    tableFormat.setBorderStyle(
        QTextFrameFormat::BorderStyle_Solid);
    tableFormat.setWidth(QTextLength(
        QTextLength::PercentageLength, 100));
```

```
m_cursor.insertTable(1, 3, tableFormat);
m_cursor.insertText(QObject::tr("Date"));
m_cursor.movePosition(QTextCursor::NextCell);
m_cursor.insertText(QObject::tr("Duration (sec)"));
m_cursor.movePosition(QTextCursor::NextCell);
m_cursor.insertText(QObject::tr("Cost"));
}

PhoneBillWriter::~PhoneBillWriter()
{
    delete m_document;
}
```

Nous commençons par ajouter une note personnelle et la faisons suivre d'un tableau et d'un en-tête de tableau, que nous remplissons avec les éléments de l'en-tête. Il n'y a pas besoin de spécifier à l'avance combien de lignes il y aura dans le tableau.

Les points intéressants concernant l'utilisation de `m_cursor`, qui inclut l'appel vers des méthodes comme **`QTextCursor::insertTable()`** et **`QTextCursor::insertText()`** afin de vraiment modifier le document.

`QTextCursor` comprend également les concepts de sélection et de suppression de texte. **`QTextCursor::movePosition()`** est une fonction très puissante du curseur, à laquelle vous pouvez passer une ou plusieurs valeurs à partir de son énumération **`QTextCursor::MoveOperation`**. Ce qui fait que la classe se comporte réellement comme un curseur dans la mesure où toutes les opérations de navigation usuelles y figurent. Dans notre cas, nous utilisons une opération de navigation introduite dans Qt 4.5 : l'opération **`QTextCursor::NextCell`** qui déplace le curseur au début de la cellule suivante du tableau. Il en résulte que le texte passé à l'appel suivant de **`QTextCursor::insertText()`** aboutira au début de la cellule.

Après avoir paramétré l'en-tête du document, nous sommes prêts à lui ajouter de réelles informations. L'appelant peut ajouter des appels téléphoniques individuels en se servant de la méthode `addPhoneCall()`, en lui passant les champs individuels que nous présentons dans le tableau.

```
void PhoneBillWriter::addPhoneCall(
    const PhoneBillWriter::PhoneCall &call)
{
    QTextTable *table = m_cursor.currentTable();
    table->appendRows(1);
    m_cursor.movePosition(QTextCursor::PreviousRow);
    m_cursor.movePosition(QTextCursor::NextCell);
    m_cursor.insertText(call.date.toString());
    m_cursor.movePosition(QTextCursor::NextCell);
    m_cursor.insertText(QString::number(call.duration));
    m_cursor.movePosition(QTextCursor::NextCell);

    QChar euro(0x20ac);
    m_cursor.insertText(QString("%1 %2").arg(euro)
        .arg(call.cost / (double) 100, 0, 'f', 2));
}
```

Pour afficher ultérieurement l'appel téléphonique, nous ajoutons une ligne à notre tableau et poursuivons en ajoutant le texte au document pour chacune des cellules du tableau. Nous appelons la méthode de **`QString`** appropriée pour convertir les données entières en texte afin de pouvoir finement calibrer la manière dont nos données seront affichées. Au final, l'objectif de notre exemple est de réaliser une version agréable à l'œil de nos données brutes, un texte convenablement formaté est le moyen d'y arriver.

Disposer juste du texte, même avec des tableaux, rend la lecture lassante. Ajouter des graphiques ou d'autres images est toujours un bon moyen de rendre un document plus attrayant. Cette page traite de la manière de créer un document ODF, et le format OpenDocument nous permet d'insérer des images dans le fichier final, contrairement à HTML, par exemple.

Dans le but d'obtenir le fichier final ODF du document, tout ce que nous avons à faire est d'introduire l'image dans le **QTextDocument**. Ce ne sera pas une grosse surprise d'apprendre qu'il y a une méthode **QTextCursor::insertImage()** qui fait exactement cela.

Pour notre exemple, nous avons voulu ajouter un graphique montrant la quantité d'appels effectués par le client dans les tout derniers mois. Pour cela, le code suivant est utilisé.

```
QList<int> callsPerMonth;
callsPerMonth << 6 << 84 << 76 << 0 << 93 << 128 << 76
               << 31 << 19 << 4 << 12 << 78;
phoneBill.addGraph(callsPerMonth, "Your past usage:");
```

Pour bien réaliser le graphique, notre solution est de créer une **QImage**, d'utiliser **QPainter** pour y dessiner les valeurs, et d'insérer l'image dans le document au bon emplacement.

```
void PhoneBillWriter::addGraph(QList<int> values, const QString &subtext)
{
    const int columnSize = 10;
    int width = values.count() * columnSize;
    int max = 0;
    foreach (int x, values)
        max = qMax(max, x);
    QImage image(width, 100, QImage::Format_Mono);
    QPainter painter(&image);
    painter.fillRect(0, 0, image.width(), image.height(),
                    Qt::white); // background
    for (int index = 0; index < values.count(); ++index)
    {
        // Adjust scale to our 100 pixel tall image:
        int height = values[index] * 100 / max;
        painter.fillRect(index * columnSize,
                        image.height() - height, columnSize, height,
                        Qt::black);
    }
    painter.end();

    QTextCursor cursor(m_document);
    cursor.movePosition(QTextCursor::End);
    cursor.insertText(subtext);
    cursor.insertBlock();
    cursor.insertImage(image);
}
```

D'abord, nous calculons la largeur voulue de notre graphique en nous basant sur la quantité de valeurs passées. Nous choisissons de toujours avoir une hauteur de 100 pixels et nous mettons les valeurs à l'échelle pour qu'elles correspondent à cet intervalle, sachant que la plupart des graphiques sont dessinés pour montrer les tailles relatives des valeurs l'une par rapport à l'autre.

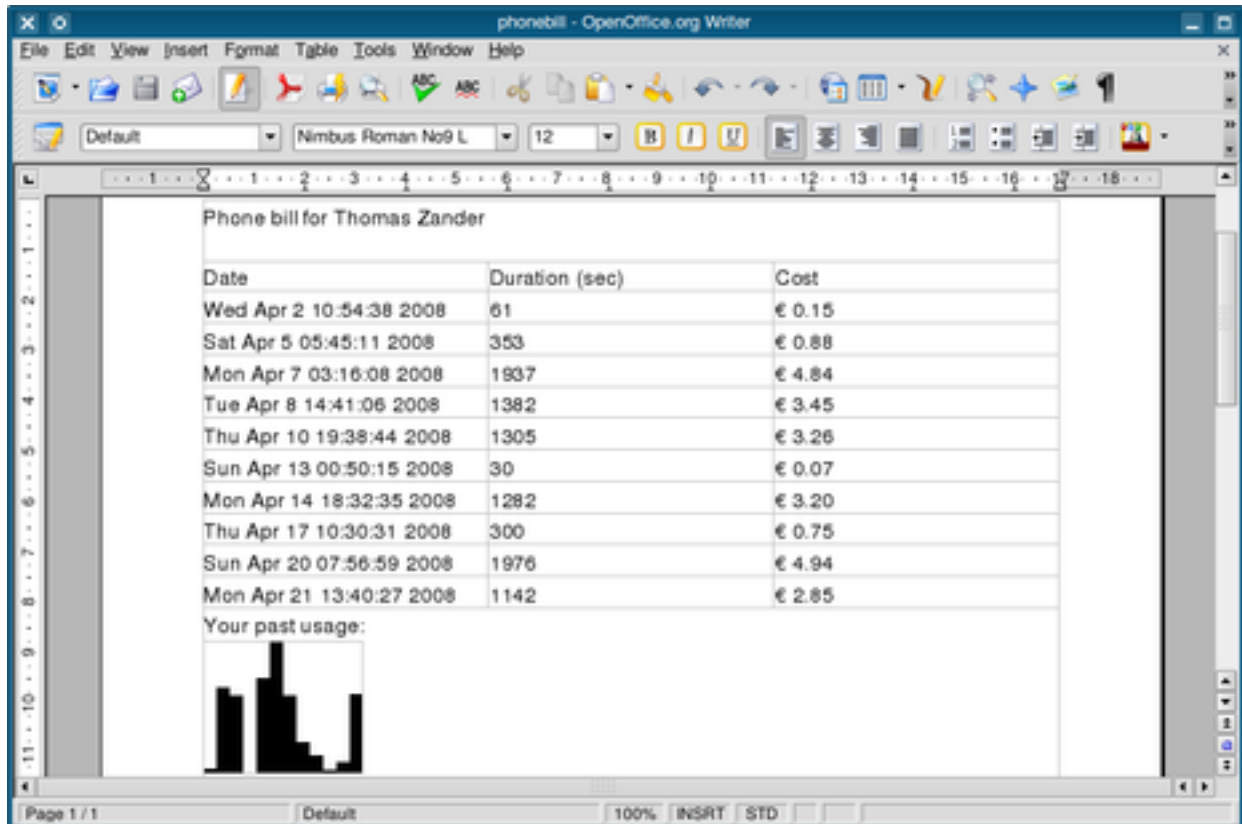
Nous utilisons un format monochrome pour l'image, ce qui signifie seulement deux couleurs. Néanmoins, vous pouvez utiliser tout format d'image que vous souhaitez dans un ODF. Deux couleurs semblent suffisantes pour le cas présent.

Après la création de l'actuelle image, nous créons un nouveau **QTextCursor** et le déplaçons à la fin du document, là où nous insérons le graphique. Une note importante : ce curseur a une position qui peut différer de l'objet `m_cursor`. La raison pour laquelle il faut créer un nouveau curseur ici est qu'il faut s'assurer que l'utilisateur peut toujours appeler `addPhoneCall()` après l'appel de `addGraph()` sans perturber le positionnement du contenu textuel.

La dernière partie de notre classe `PhoneBillWriter` sera de créer le fichier au format OpenDocument à partir de toutes les informations que nous avons mises dans l'objet de saisie. Le plus gros travail est fait par la classe **QTextDocumentWriter**, qui est capable d'écrire vers des formats variés, fichier texte, HTML et ODF. Il sort en ODF par défaut, rendant l'implémentation triviale pour nous.

```
void PhoneBillWriter::write(const QString &fileName)
{
```

```
QTextDocumentWriter writer(fileName);
writer.write(m_document);
}
```



L'image ci-dessus montre le document final dans OpenOffice.org. Tel quel, il est intentionnellement simple dans le but de la démonstration, mais il s'en trouve un exemple sensiblement plus décoratif inclus dans l'archive.

V - Conclusions

L'exemple que nous avons présenté est assez simple et s'est davantage focalisé sur les capacités de base de **QTextDocumentWriter** pour écrire de l'ODF que sur le fait de réaliser la plus jolie présentation possible.

Qt possède une multitude de mécanismes qui peuvent être utilisés pour améliorer l'apparence de la note téléphonique, et utiles à la saisie. Par exemple, nous pouvons utiliser le module **Qt SQL** pour extraire de véritables informations d'une base de données existante et utiliser **QPainter** pour dessiner de plaisants graphiques et diagrammes.

Nous pouvons également créer une jolie interface graphique pour aider l'utilisateur à créer ses rapports ou utiliser les capacités de Qt pour ODF afin de réaliser quelque chose de complètement sans rapport avec une note téléphonique, des rapports ou de la comptabilité. Ce que vous créerez avec ce nouveau dispositif ne dépendra que de vous !

VI - Divers

J'adresse ici de chaleureux remerciements à **Alp, yan, dourouc05** pour leurs encouragements, leurs relectures et corrections, et à **Baptiste Wicht** pour sa relecture orthographique !

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 !

Source **Le code source de l'exemple**