Qt ET XML

Présentation

Le format XML connaît aujourd'hui une popularité grandissante : lisible par tous, il permet l'échange de données entre machines ou entre applications (QtDesigner sauvegarde son fichier xxx.ui au format XML). Pour écrire ou lire un document XML, il existe différentes méthodes. Pour illustrer ce tutoriel, on utilisera le document XML suivant :

```
<?xml version="1.0" ?>
<mesures>
	<mesure numero="1">
		<tension>10</tension>
		<frequence>1000</frequence>
	</mesure>
	<mesure numero="2">
		<tension>20</tension>
		<frequence>2000</frequence>
	</mesure>
</mesures>
```

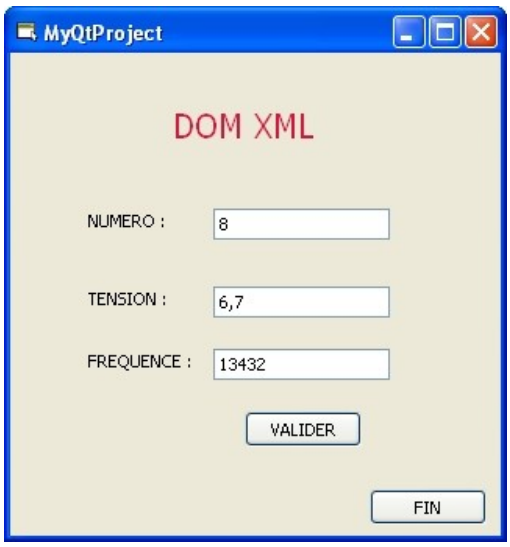
Ecriture d'un document XML avec la méthode DOM

Introduction à DOM

Un document XML peut être schématisé comme une arborescence hiérarchisée. Les différentes composantes d'une telle arborescence sont désignées comme étant des noeuds. Il existe différents types de nœuds : les nœuds éléments, les nœuds attributs, les nœuds textes, etc. Le nœud élément correspond à la balise, le nœud attribut à l'attribut qu'on peut trouver dans la balise ouvrante et le nœud texte à la donnée proprement dite. Dans le document XML ci-dessus, "mesure" est un nœud élément, "numero" un nœud attribut et "10" un nœud texte. On doit établir une filiation pour les nœuds éléments et les nœuds textes : ici, "10" est enfant de "tension", lui-même enfant de "mesure", lui-même enfant de "mesures", lui-même enfant du document Dom.

Présentation de l'application

On souhaite créer un fichier "mesures.xml" suivant le format décrit ci-dessus contenant des données saisies par un utilisateur. L'IHM aura l'aspect ci-contre.



Le document XML est une instance de QDomDocument. Les nœuds sont des instances de QDomElement ou QDomText.

La méthode createElement("nom_balise") du document crée un nœud « élément » (on lui passe le nom de la balise).
La méthode setAttribute("numero", n) permet de créer un attribut "numero" avec la valeur n.
La méthode createTextNode("donnees") du document crée un nœud « texte » (on lui passe la donnée).
La filiation des différents nœuds se fait comme suit : parent.appendChild(enfant) ;

Codage

La classe EcritureDom dérive de QObject car elle contient un slot publique demande_ajout(QString,QString,QString). Ce slot, connecté à l'IHM, reçoit en paramètres les 3 données saisies par l'utilisateur.

```
class EcritureDom : public QObject
{
    Q_OBJECT
public:
    EcritureDom();
    ~EcritureDom();
public slots :
    void demande_ajout(QString,QString,QString);
private :
    QDomDocument doc;
    QDomElement mesures;
    QFile file;
    QTextStream out;
};

EcritureDom::EcritureDom()
{
    mesures = doc.createElement("mesures");   //creation de la balise “mesures”
    doc.appendChild(mesures);                 //filiation de la balise “mesures”
    file.setFileName("mesures.xml");
    if (!file.open(QIODevice::WriteOnly))     //ouverture du fichier de sauvegarde
        return;                               //en ecriture
    out.setDevice(&file);                     //association du flux au fichier
}
```

```
void EcritureDom::demande_ajout(QString n,QString t,QString f)
{
    //creation de la balise “mesure”
    QDomElement mesure = doc.createElement("mesure");
    mesures.appendChild(mesure);        //filiation de la balise “mesure”
    mesure.setAttribute("numero",n);    //creation de l’attribut “numero”

    //creation de la balise “tension”
    QDomElement tension = doc.createElement("tension");
    mesure.appendChild(tension);        //filiation de la balise “tension”

    //creation de la balise “frequence”
    QDomElement frequence = doc.createElement("frequence");
    mesure.appendChild(frequence);              //filiation de la balise “frequence”
    QDomText t1 = doc.createTextNode(t);        //création de la donnée t1
    tension.appendChild(t1);                    //filiation du noeud “t1”
    QDomText f1 = doc.createTextNode(f);        //création de la donnée f1
    frequence.appendChild(f1);                  //filiation du noeud “f1”
}

EcritureDom::~EcritureDom()
{
    //insertion en début de document de <?xml version="1.0" ?>
    QDomNode noeud = doc.createProcessingInstruction("xml","version=\"1.0\"");
    doc.insertBefore(noeud,doc.firstChild());

    //sauvegarde dans le flux (2 espaces de décalage dans l’arborescence)
    doc.save(out,2);
    file.close();
}
```

Lecture d’un document XML avec la méthode DOM

Présentation de l’application

On cherche cette fois à lire le fichier « mesures.xml » contenant les données précédemment présentées. L’application ne fait ici que présenter les données « mesure par mesure » dans une QMessageBox, mais il est évident que les données récupérées peuvent être envoyées dans un signal à destination d’une classe de traitement ou vers une IHM.
Le principe est assez simple. Une fois le document DOM créé et son arborescence établie (la méthode setContent fait tout le boulot), on passe en revue les nœuds « mesure » un par un. Pour chaque nœud, on mémorise l’attribut et on crée une liste d’enfants (ici, tension et fréquence). Il ne reste plus qu’à rechercher la donnée texte pour chaque enfant.

Codage

La classe Lecture_DOM dérive de QObject car elle contient un slot publique lire( ). Ce slot est appelé pour débuter la lecture.

```
class Lecture_DOM : public QObject
{
    Q_OBJECT
public:
    Lecture_DOM();
    ~Lecture_DOM();
public slots :
    void lire();
private :
    QDomDocument doc;
};

Lecture_DOM::Lecture_DOM()
{
    QFile file("mesures.xml");
    if (!file.open(QIODevice::ReadOnly))
        return;
    if (!doc.setContent(&file)) {              //établit le document XML à
        file.close();                          //partir des données du fichier (hiérarchie, etc)
        return;
    }
    file.close();
}

void Lecture_DOM::lire()
{
    int i=0;
    QString affichage;
    QDomNodeList tab;
    QDomElement mesure;
    QDomNode n;
    QMessageBox a(0);
    QDomElement racine = doc.documentElement();      //renvoie la balise racine
    QDomNode noeud = racine.firstChild();        //renvoie la 1ère balise « mesure »
    while(!noeud.isNull())
    {
        //convertit le nœud en élément pour utiliser les
        //méthodes tagName() et attribute()
        mesure = noeud.toElement();
```

```
        //vérifie la présence de la balise « mesure »
        if (mesure.tagName() == "mesure")
        {
            affichage = mesure.attribute("numero");          //récupère l'attribut
            tab = mesure.childNodes();     //crée un tableau des enfants de « mesure »
            for(i=0;i<tab.length();i++)
            {
                //pour chaque enfant, on extrait la donnée et on concatène
                n = tab.item(i);
                affichage = affichage + " " + n.firstChild().toText().data();
            }
            a.setText(affichage);              //affichage dans un QMessageBox
            a.exec();
        }
        noeud = noeud.nextSibling();       //passe à la "mesure" suivante
    }
}
```

Lecture d'un document XML avec la méthode SAX

Introduction à SAX

Le modèle SAX consiste à parcourir le document linéairement en une seule fois et à déclencher des méthodes à chaque fois qu'une des catégories syntaxiques (balise ouvrante, fermante, texte, etc) est rencontrée.

Présentation de l'application

Le but recherché est le même qu'avec la méthode DOM : on souhaite présenter les données « mesure par mesure » dans une QMessageBox.
Pour extraire les données d'un fichier XML, on dérive la classe QXmlDefaultHandler et on implémente les méthodes permettant la gestion des balises ouvrantes, fermantes, des données et des erreurs.

```
class SaxXml : public QXmlDefaultHandler
{
public :
    SaxXml() ;
    virtual  bool startElement(
                const QString &namespaceURI,    //inutile ici
                const QString &localName,       //inutile ici
                const QString &qName,           //nom de la balise
                const QXmlAttributes &attribs) ;    //liste des attributs
```

```
    virtual bool endElement (
                const QString &namespaceURI,     //inutile ici
                const QString &localName,        //inutile ici
                const QString &qName) ;          //nom de la balise
    virtual bool characters (const QString &str );
    virtual bool fatalError(const QXmlParseException &e) ;
};
```

Les méthodes retournent « true » pour continuer l 'analyse du document.

Pour parser le document (l 'analyser), il faut commencer comme suit :

```
QFile file;
QXmlInputSource *inputSource;
QXmlSimpleReader reader;                    //une interface pour notre parseur
SaxXml handler;                             //notre classe qui va faire le boulot
file.setFileName("mesures.xml");            //spécifie le nom du fichier xml à lire
inputSource= new QXmlInputSource(&file);    //associe une source xml au fichier
reader.setContentHandler(&handler);         //associe l'interface à notre parseur
bool ok=reader.parse(inputSource);          //débute la lecture du document xml
if (!ok) {
    QMessageBox a(0);
    a.setText("problem");
    a.exec();
}
```

Codage

La classe SaxXml dérive de QObject pour pouvoir, si nécessaire, envoyer des signaux. Elle dérive aussi de QXmlDefaultHandler, classe nécessaire à QXmlSimpleReader. Les méthodes virtuelles devront être implémentées : c'est là que se trouve le code relatif à notre application.

Le principe est simple. Quand une balise « ouvrante » est découverte par le parseur, la méthode startElement est appelée. Si le nom de la balise est "mesure", on va chercher son attribut "numero" et on positionne un drapeau afin de mémoriser ce passage. On mémorisera également les passages dans les balises ouvrantes nommées "tension" ou "frequence". Dans la méthode "characters" appelée lorsque le parseur rencontre des données, on teste les balises pour savoir quoi faire de ces données (ici, on les concatène). Dans la méthode "endElement" appelée lorsque le parseur rencontre une balise fermante, on réinitialise les drapeaux.

```
class SaxXml  :public QObject, public QXmlDefaultHandler
{
    Q_OBJECT
public:
    SaxXml();

    virtual bool fatalError (const QXmlParseException & exception);
    virtual bool characters ( const QString &);              //traite les données
    virtual bool endDocument ();                             //traite la fin du document
    virtual bool endElement (                                //traite la balise fermante
            const QString &, const QString &, const QString &  );

    virtual bool startDocument () ;                          //traite le début du document
    virtual bool startElement ( const QString &,             //traite la balise fermante
            const QString &, const QString &, const QXmlAttributes & );

private :
    QString balise_mesure;       //drapeau pour une balise “mesure”
    QString balise;              //drapeau pour une balise “tension” ou “frequence”
    QString affichage;           //pour la QMessageBox
};

SaxXml::SaxXml() { }    //constructeur

bool SaxXml::startDocument ()
{
    return true;             //pas traité ici
}

bool SaxXml::startElement ( const QString & , const QString & , const QString & qName,
const QXmlAttributes & atts )
{
    if(qName=="mesure")
    {
        balise_mesure = "mesure";         //drapeau "mesure" pour mémorisation
        affichage = atts.value(0) + " ";      //mémorisation de l’attribut pour QMessageBox
    }
    if (qName=="frequence" || qName=="tension")
        balise=qName;              //drapeau "frequence" ou "tension" pour mémorisation
    return true;
}
```

```
bool SaxXml::fatalError (const QXmlParseException & exception)
{
    return false;              //pas traité ici
}

bool SaxXml::characters ( const QString & ch)
{
    if (balise_mesure=="mesure" && !ch.isEmpty())        //test du drapeau "mesure"
    {
        if(balise=="frequence" || balise=="tension")          //test des autres drapeaux
            affichage = affichage + ch + " ";          //concaténation pour QMessageBox
    }
    return true;
}

bool SaxXml::endDocument ()
{
    return true;
}

bool SaxXml::endElement ( const QString & namespaceURI, const QString &
localName, const QString & qName )
{
    if (qName=="mesure")
    {
        balise_mesure=" ";          balise=" ";        //réinitialisation des balises
        QMessageBox a(0);
        a.setText(affichage);
        a.exec();
        affichage = " ";                                //réinitialisation de la chaîne
    }
    return true;
}
```