How-to: parsing XML with Qt

Why use XML?

The purpose of this document is to show a couple ways of transforming data from a saved state into useful objects in memory, on the Boardcon 2440. The document type best suited to this task is XML, and as the Boardcon 2440 mark III sports a Qt-everywhere implementation, which includes QTs XML parsing objects, parsing XML with QT will be the subject of this paper.

The advantage of XML over a regular csv file, or custom character or binary file is that "records" or "elements" can be different from one another while still being parsed by the same code.

In designing a trivia game, if you decided you needed a `Question` object, which contained question text, several different answer texts and an indication of which answer is correct, and you decided to do this with a csv file, your data might look like this:

```
1, What color is the sky?, red, green, blue, purple, 2,
2, What is the air speed velocity of an African swallow? 24mph, 10 mph, 5Mm/h, blue, 0
3, What's Monty Python? An English comedy group, a particularly scary snake, A
dishwasher brand, blue, 0
```

… using a file stream reader and tokenizing on strings with a comma "," as the delimiter.

But this is problematic , as adding any of the following would require lots of additional development:

- Support for questions with several answers
- Support for questions with external references or resources
- Support for comments

Using the XML alleviates most of these problems.

```
<Question ID="1" CorrectAnsID="3" >
      <QuestionText>What color is the sky?</QuestionText>
      <Answer ID="1">red</Answer>
      <Answer ID="2">green</Answer>
      <Answer ID="3">Blue</Answer>
      <Answer ID="4">Purple</Answer>
</Question>
<Question ID="2" CorrectAnsID="1" >
      <QuestionText> What is the air speed velocity of an African
swallow?</QuestionText>
      <Answer ID="1">24mph</Answer>
      <Answer ID="2">5mph</Answer>
```

```
      <Answer ID=”3”>5Mm/h</Answer>
      <!-- that’s Mega-meters per second, and this is a comment, which
the parser will automatically handle for us.-->
      <Answer ID=”4”>Purple</Answer>
</Question>
<!-- below is an example of some functionality that could be added,
even though this snippit is still valid XML and could conceivably be
used by the same code -->
<Question ID=”2” CorrectAnsID=”1” Type=”MultiMedia”>
      <Resource Type=”AUDIO”>monty-airspeed.mp3</Resource>
      <Resource Type=”IMAGE”>monty-swallow.jpg</Resource>
      <QuestionText> What’s Monty Python ? </QuestionText>
      <Answer ID=”1”>24mph</Answer>
      <Answer ID=”2”>5mph</Answer>
      <Answer ID=”3”>5Mm/h</Answer>
      <!-- that’s Mega-meters per second, and this is a comment -->
      <Answer ID=”4”>Purple</Answer>
</Question>
```

Qt is a sophisticated high-level framework on top of C++, as such it provides several objects to access elements embedded in an XML file. It includes a complete implementation as per the W3C’s Document Object Model (“DOM”) called QDomDocument, but it also includes an easier to use QXmlStreamReader, which abstracts away from the sibling tree-like nature of XML and just gives you a serialized view.

Writing your XML document

General resources:

- On linux, gedit provides basic XML highlighting if the extension is xml.
- On windows, notepad++ can be used to do the same.
- The XML plugin to eclipse does not provide highlighting, it abstracts further, into a 2-column table with drop down menus.
- XML validators. Use this to ensure your XML document is valid:
    - http://www.w3schools.com/xml/xml_validator.asp

Writing the Qt XML parsing code

Open a terminal, and change to your projects directory:

```
>cd  ~/cmpt433/YourUserName/private/myApps/YourProject
```

Adding support for XML to your project

First, modify your projects .pro to indicate to the QT pre-compiler that you'll need XML parsing objects:

```
>gedit ./yourProject.pro
#-------------------------------------------------
#
# Project created by QtCreator 2011-11-12T15:31:45
#
#-------------------------------------------------

QT       += core gui xml

TARGET = YourProject
...
```

Including the needed Libraries

Include the libraries "QXmlStreamReader ", "QFile", and "QDir" in the header file of the object that's going to be reading the XML. You may not need the latter two, but they are very helpful.

In your code editor of choice, or with gedit:

```
 >gedit ./yourXMLParsingObject.h
#include <QObject>
#include <QDebug>
#include <QXmlStreamReader>
#include <QFile>
#include <QDir>
class YourXMLParsingObject ...
```

Get the XML file to parse

Navigating directories can be tricky, though it is not the subject of this how-to, the QDir and QFile objects can aide in this. If your application can be thought of as a shell then a QDir object can be thought of as the present-working-directory. Some helpful methods to note:

- QDir.AbsolutePath() gets the fully qualified path to the directory its currently examining.
- QDir.cdUp() moves the QDir object up one level in the directory tree
- QDir.cd(Qstring dirName) moves the QDir down one level in the directory tree to the specified sub-directory

To allow your application to run on both your target, and any windows or linux development environments, It's also helpful to use a constant static field or a #define to hold the path of the file, and a precompiler switch to determine where the file is in your directory structure. Modify `yourXMLParsingObject.h` again to include this statement:

```
class Model ...

private:
      const QString xmlFilePath;
...
```

Followed by modifying the `yourXMLParsingObject.cpp` file to set that variable, machine dependent. Unfortunately, because of the nature of C++'s handling of constant object field references this is going to get ugly, but it does give the desired effect:

```
#include "model.h"

//constructor
yourXMLParsingObject::yourXMLParsingObject(
            QObject *parent) :
      QObject(parent), AnyDependentObjects(),
      #ifdef __arm__ //on the target
            xmlFilePath("/mnt/remote/YourProject/myXML.xml")
      #else
      #ifdef __WIN32 //for those developing on windows
            xmlFilePath("C:\\Users\\You\\QT\\YourProject\\myXML.xml")
      #else //for those developing on linux
            xmlFilePath("/home/You/QT/YourProject/myXML.xml")
      #endif
      #endif
{
    score = 0;
    lives = 4;
      ...
```

Some notes

- Because Qt on Windows uses mingw and some ported linux libraries, finding a defined constant to use as a precompiler `#ifdef` constant is difficult. As of writing, I have not found one.
- If you wish to add further dependencies after this one, put a comma after each of the xmlfilePath constructor calls (read: turn "`xmlFilePath(…)`" into "`xmlFilePath(…),`" for each of them) then any additional dependencies you need after the last `#endif`.
- Using the non-escape character for a file path on windows (IE using "/" instead of "\\") will work, but windows traditionally handles file paths with a "\" rather than a "/", so convention is why "\" is used here.

Parsing the XML file

Given the XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Quiz>
	<GlobalVar ID="ODE2C0DE">cool Value with Digits 2.45</GlobalVar>
	<QuestionSet>
		<Question ID="1">
			<Text>What is your favorite Color?</Text>
			<Answer>Blue</Answer>
		</Question>
		<Question ID="2">
			<Text>How much Do I love QT</Text>
			<Answer>At 10:34PM on Friday, not a whole lot</Answer>
		</Question>
	</QuestionSet>
</Quiz>
```

Below is known working code with comments explaining the behavior:

```
//Assuming somewhere we have a question object, similar to:
Struct GlobalVarADT{
	int ID;
	QString text;
GlobalVarADT globalVar;

Struct Question{
	int questionID;
	QString questionText;
	QString answerText;
}
QList<Question> questions;

//This is going to be a member function of yourXMLParsingObject
//that takes a filePath and reads each "question" element into a
//QList of question objects.
void YourXMLParsingObject::yourXMLParsingMethod(QString filePath){

	//create a new file object with the XML file path.
	QFile* file = new QFile(xmlFilePath);

	//test to see if the file is readable and that it contains text.
	if(!file->open(QIODevice::ReadOnly | QIODevice::Text)){
		//if you wish to perform some action if the file is inaccessible,
```

```
	//do so here.
	return;
}

//Create the XML stream reader.
QXmlStreamReader xml(file);

//we're going to loop over the entire xml document
//using QXmlStreamReader's atEnd() method, in addition to
//its hasError() method
while(!xml.atEnd() && !xml.hasError()){

	//read the next piece of data into the reader.
	//this will move the reader "over" or "onto" the next
	//valid element, including any attributes it holds as
	//well as its value. The returned object is a status object
	//which can be tested for EOF, the begging of the file, etc.
	QXmlStreamReader::TokenType token = xml.readNext();

	//test to see if the token indicates that we're at the beginning
	//of the document. Any calls to the reader's current data will
	//give you the xml version number information.
	if(token == QXmlStreamReader::StartDocument){
		//we don't want any of this data, it isn't any element
		//we need.
		continue;
	}

	//what we're looking for is that start of a valid element
	if(token == QXmlStreamReader::StartElement) {

	   //any global elements or elements of quiz that you wish
	   //to read in come next
	   if(xml.name() == "GlobalVar"){

			//here we've got globalVar, so I'm going to
			//save the ID attribute, and save its text to the
			//globalVar
			globalVar.ID =
				xml.attributes().value("ID").toString().toInt();
			//So your XML reader was looking at the GlobalVar
			//element, we need to push it forward (such that its
			//looking at the value field) so that it reads the
			//text of this current element.
			xml.readNext();

			//now save the xml.text(), this will be the value in
			//the globalvar element (<GlobalVar>HERE</GlobalVar>)
			globalVar.text = xml.text().toString();
	   }

	   //Questionset itself doesn't do anything for us. We need its
```

```
                //children, so just skip this element
                if(xml.name() == QuestionSet){
                        continue;
                }

                //This is a question, in the name of good software
                //engineering I'm deligating this off to another
                //function called "parseQuestion".
                if(xml.name() == "Question") {
                        this->parseQuestion(xml);
                }
            }//startElement
        }//while
}//function

void YourXMLParsingObject::parseQuestion(QXmlStreamReader& xml){
        //check to ensure that we were called in the appropriate spot!
        if(xml.tokenType() != QXmlStreamReader::StartElement
                        && xml.name() != "Question"){
                qDebug() << "Called XML parseQuestionElement "
                        << "without a question Element in the XML stream!";
                return;
        }

        //create some memory to read the fields into
        Question* newQuestion = new Question();

        //read the attribute fields in first: You know that currently the
        //stream is pointing at the Question element (as was checked by
        //the previous if condition), so you can read in the attribute
        //fields immediately.

        newQuestion->ID = xml.attributes().value("ID").toString().toInt();

        //but now we need to get data from Question's children...
        xml.readNext();

        //and now the code looks very similar to that of the caller!
        //the way to read the while condition is as follows:
        //while it isn't the case that we're at the end of an element and
        //(to protect against nested questions) we're not looking at a
        //new question element
        while(!(xml.tokenType() == QXmlStreamReader::EndElement
                && xml.name() == "Question")){

                //at the start of an element, otherwise ignore and
                //keep reading.
                if(xml.tokenType() == QXmlStreamReader::StartElement){

                        //If the element is a text element, save it
                        if(xml.name() == "Text"){
                                xml.readNext();
                                newQuestion->questionText =
                                        xml.text().toString();
                        }
                        //otherwise if its an answer element, save it
```

```
                        if(xml.name() == "Answer"){
                                xml.readNext();
                                newQuestion->answerText =
                                        xml.text().toString();
                        }
                        //otherwise we don't know what it is, so do nothing,
                        //and read the next thing!
                }
                //we're currently looking at some element feld,
                //so read the next element header!
                xml.readNext();
        }//while

        //add the new question we just created to the list.
        questions.add(newQuestion);

        //done!
        return;
}
```