Putting XML to Work in Your Application with XQuery

Abstract

The use of XML as way to share structured data across information systems on- and offline is increasing, however effectively using XML data in your application can be cumbersome, demanding many lines of complex code. This whitepaper discusses how to put XML and other structured data to work in your application with XQuery – the standard language for querying XML data, – vastly simplifying the use of XML in the Qt application framework.

NOKIA

Introduction

XQuery[1] is a query language for traversing XML documents to select and aggregate XML elements of interest and transform them for output as XML or in some other format. XPath is the element selection part of XQuery. XQuery simplifies query tasks by eliminating the need for large amounts of procedural programming in C++ or Java. Consider a simple but powerful XQuery:

```
declare namespace db = “http://www.docbook.org/ns/docbook”;
doc('ftp://ftp.example.com/example.xml')//p/<db:para>{node()}</db:para>
```

This query finds each *p* element in the file *example.xml*, creates a new *para* element to replace the *p* element (where *para* is defined in the *docbook* namespace), and then copies all the *p* element's child elements into the *para* element.[2] Imagine the equivalent C++ or Java code you would have to write to do the same work. In XQuery, it's essentially a one-liner – a namespace declaration, followed by a single instruction.

The cross-platform application framework Qt[3] version 4.4 introduces a new module called **QtXmlPatterns** that supports XQuery 1.0 and XPath 2.0 in Qt programs. This document is an introduction to QtXmlPatterns.

QtXmlPatterns Makes Running XQueries Simple

How can we run the XQuery shown above from a Qt program? There is a standard pattern for running an XQuery in QtXmlPatterns, with several variations depending on where the XQuery text comes from and where the XQuery output goes to.

The simplest is to input the XQuery to QtXmlPatterns in a QString and direct the output to a QIODevice formatted as XML.

```
QString  xq = “declare namespace db = \“http://www.docbook.org/ns/docbook \”” +
               “doc('ftp://ftp.example.com/example.xml')//p/<db:para>{node()}</db:para>”;
QXmlQuery query;
query.setQuery(xq);
QXmlFormatter formatter(query, myOutputDevice);
query.evaluateTo(&formatter);
```

The snippet above shows the standard sequence for running an XQuery in QtXmlPatterns:

1. Create a query object (QXmlQuery).
2. Initialize the query's XQuery (QXmlQuery::setQuery()).
3. Set up the query's output stream (QXmlFormatter).
4. Evaluate the query (QXmlQuery::evaluateTo()).

You can vary the source of the XQuery passed to QXmlQuery::setQuery() in step 2. The XQuery can be read from a file or from an input device, or it can be loaded from a URL. You also can vary the destination for the query result we establish in step 3 or 4. You can write the output to a file or device formatted as XML, as in the code sample, or we can store the output in a list of XML items (QXmlResultItems) or strings (QStringList) for further processing. You can even send the output to another XML processor directly (QAbstractXmlReceiver).

1 XQuery was designed by the W3C, http://www.w3.org.
2 Hint: The example query converts html to docbook.
3 Qt is a multi-platform C++ GUI toolkit that provides application developers with all the functionality for building applications with state-of-the-art graphical user interfaces. Qt is object-oriented, extensible, and allows true component programming. Qt is available under both proprietary and open source licensing.

Using XQuery Versus DOM-based code

Procedural code that manipulates the XML Document Object Model (DOM) often is used where running an XQuery in QtXmlPatterns would be much simpler. Even a solution based on Qt's QDom and QtNetwork classes can be replaced by simpler use of XQuery and QtXmlPatterns. Consider the following DOM-based algorithm. It loads the document at http://example.com, iterates over all the children of the document's *root* element to find the *p* element with class attribute *a54*, returning the element as a string.

```
QNetworkAccessManager manager;
QNetworkReply *const reply = manager.get(“http://example.com/”);
reply->waitForReadyRead();
QBuffer content;
content.setData(reply->readAll());
QDomDocument doc;
content.open(QIODevice::ReadOnly);
doc.setContent(&content);
QDomElement docEle = doc.documentElement();
if  (docEle.name() != “root”)
            return;
QString result;
for (int i = 0;  i < docEle.childNodes().count(); ++i) {
            QDomNode n = docEle.childNodes().at(i);
            if (n.nodeType() == QDom::Element && n.name() == “p”) {
                        if (n.toElement().attribute(“class”) != “a54”)
                                    continue;
                        QDomChildNodes children = n.childNodes();
                        for (int c = 0; c < children.count(); ++c) {
                                    if (children.at(c).nodeType() == QDom::TextNode)
                                                result.append(children.at(c).text();
                        }
                        break;
            }
}
delete reply;
return result;

Here is an XQuery that does the same thing:

            doc(“http://example.com”)/root/p[@class='a54']/string()

You can use the standard pattern to evaluate the query in C++:

QXmlQuery query;
query.setQuery(“doc('http://example.com/')/root/p[@class='a54']/string()”);
QStringList result;
query.evaluateTo(&result);
```

Evaluating an XQuery in QtXmlPatterns is much simpler, but if this example is not dramatic enough to make you want to abandon the DOM-based approach, consider a more complex scenario. Suppose that you also want to generate a new element and copy all the attributes of the old element into the new element, except the *class* attribute. Using XQuery and QtXmlPatterns, all you need do is modify the XQuery:

```
doc('http://example.com/')/root/p[@class='a54']/<newElement>{@* except @class}</newElement>
```

The C++ code to evaluate the XQuery remains the same. But modifying the DOM-based algorithm to accomplish the additional task results in a code snippet that is too long to show here.

Query any hierarchical data (not just XML)

Most use cases for QtXmlPatterns will involve running XQueries on XML data using the standard C++ pattern. But the QtXmlPatterns module also can be used to run XQueries on *any* hierarchical data that can be modeled to look like XML. Conversion of your custom data to XML format is neither required nor recommended. Your XQuery traverses the hierarchical data directly, as if it were XML, selecting and aggregating the desired elements, and then outputting them as XML or making them available for further C++ or Java processing via the QtXmlPatterns APIs already presented.

To use XQuery on non-XML data, you first model it as an XML tree by writing a subclass of the QtXmlPatterns base class **QAbstractXmlNodeModel**. Consider the *File System Example* in the Qt documentation.[4] It subclasses QAbstractXmlNodeModel to make class FileTree, which models the local computer's file system as an XML tree. Class FileTree uses Qt's QFileInfo class internally to interpret the file system's directory and file structure and make it traversable by the XQuery engine.

Simplifying standards-based computing

XML and associated specifications abound, but they are often difficult to understand fully and to apply in detail. Consider these issues that arise when writing C++ code to handle XML:

- Should empty default namespaces be declared?
- Should *CDATA* sections be dealt with differently from regular text nodes?
- Is *nbsp* a built-in entity in XML?

Writing C++ code to handle these details and a myriad of others is a process typically fraught with subtle bugs. A large part of the work done by QtXmlPatterns consists of validating and checking code to ensure that XQuery output will be fit for consumption by third-party XML tools. The details handled by QtXmlPatterns include:

- Generation of correct namespace declarations as required. Namespace declarations must be inserted manually by DOM-based code.
- Validation of data node content to ensure it will produce well-formed XML.
- Automatic error reporting, whenever data errors are encountered.

Because all this is handled transparently, developers can concentrate on the actual data processing problem without worrying about conformance to specifications.

4 See the File System Example http://doc.trolltech.com/4.4rc1/examples.html#xml-patterns-xquery-xpath or later release.

Technical information and future development

Qt's XQuery support is delivered via a C++ library, Java bindings (via QtJambi), and a command line tool for commandline scripting. The implementation currently passes 97% of W3C's XQuery test suite.

Support for XSL-T 2.0 is currently under development and is expected to be included in Qt 4.5. The XProc processing flow language and the XQuery Update Facility are being considered for inclusion in a future release.

Key Points

- QtXmlPatterns fully supports the use of XQuery 1.0 and XPath 2.0 in Qt programs for traversing XML documents to select and aggregate data of interest for transformation and output as XML or some other format.
- QtXmlPatterns enhances the capabilities of XQuery and XPath by allowing them to operate on non-XML, hierarchical data that has been modeled to look like XML.
- QtXmlPatterns provides the tools the user needs to create these models.
- QtXmlPatterns currently passes 97% of W3C's XQuery test suite.

Further reading

More sources of information about Qt's XML:

- Online Qt documentation: http://doc.trolltech.com/main-snapshot/qtxmlpatterns.html
- Download Qt from http://www.trolltech.com and use the bundled documentation tool *Assistant* to read the documentation, and run the QtXmlPatterns demo applications.
- Qt Labs: http://labs.trolltech.com/

About Qt Software

Qt Software (formerly Trolltech) is a global leader in cross-platform application frameworks. Nokia acquired Trolltech in June 2008, renamed it to Qt Software as a group within Nokia. Qt allows open source and commercial customers to code less, create more and deploy everywhere. Qt enables developers to build innovative services and applications once and then extend the innovation across all major desktop, mobile and other embedded platforms without rewriting the code. Qt is also used by multiple leading consumer electronics vendors to create advanced user interfaces for Linux devices. Qt underpins Nokia strategy to develop a wide range of products and experiences that people can fall in love with.

About Nokia

Nokia is the world leader in mobility, driving the transformation and growth of the converging Internet and communications industries. We make a wide range of mobile devices with services and software that enable people to experience music, navigation, video, television, imaging, games, business mobility and more. Developing and growing our offering of consumer Internet services, as well as our enterprise solutions and software, is a key area of focus. We also provide equipment, solutions and services for communications networks through Nokia Siemens Networks.

Code less.
Create more.
Deploy everywhere.

NOKIA

Oslo, Norway +47 21 60 48 00 ■ **Redwood City, CA, USA** +1 650-551-1676 ■ **Beijing, China** +86 10 6588 8836 ■ qt-info@nokia.com **www.trolltech.com**