XML in .NET

Source http://www.xml.com/lpt/a/932

In his opening keynote at the IDEAlliance XML 2001 Conference in Orlando, Florida, in December, James Clark said: "Just because it comes from Microsoft, it's not necessarily bad". With that in mind, I decided to explore what C# has to offer to the Java-XML community.

I've been watching the continuing Microsoft story with a vague combination of intrigue and apprehension. You almost certainly know by now that, due to an awkward combination of hubris and court orders, Microsoft has stopped shipping any Java implementation with Windows, choosing instead to hitch its wagon to a star of its own making, C#.

As a consumer, I'm not sure whether I like Microsoft's business practices. As a software developer, however, I'm interested in learning new languages and technologies. I've read enough to see that C# is enough like Java to make an interesting porting project. Even if I never write another line of C# code, there is certainly a lot to be learned from how Microsoft has integrated XML into its .NET platform.

In this series I'll be porting a few small XML applications, which I've hypothetically written in Java, to C# in order to see if I can improve my Java programming.

The Exercises

The first Java application to port to C#, which I call RSSReader, does something that most XML programmers have done at some point: read in an RSS stream using SAX and convert it to HTML. For our purposes, I'll expect to be reading an RSS 1.0 stream using JAXP and outputing out an HTML stream using java.io classes. We'll see that this example ports nicely to the C# XmlReader class.

Future examples will convert JDOM to the C# XmlDocument and XmlNode classes, as well as experimenting with ports from an XML databinding framework to ADO.NET. There's a lot to ADO.NET, and I'll discuss some of that as well.

Here's our first Java program, RSSReader, which was adapted from Sun's JAXP Tutorial. I've stripped out some of the error handling and such for the sake of simplicity.

```
package com.xml;

import java.io.*;
import java.util.Stack;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

public class RSSReader extends DefaultHandler {
```

```
static private Writer out;
static String lineEnd =  System.getProperty("line.separator");

Stack stack = new Stack();
StringBuffer value = null;
String title = null;
String link = null;
String desc = null;

public static void main(String args []) {
  // create an instance of RSSReader
  DefaultHandler handler = new RSSReader();

  try {
    // Set up output stream
    out = new OutputStreamWriter(System.out, "UTF8");

    // get a SAX parser from the factory
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser saxParser = factory.newSAXParser();

    // parse the document from the parameter
    saxParser.parse(args[0], handler);

  } catch (Exception t) {
    System.err.println(t.getClass().getName());
    t.printStackTrace(System.err);
  }
}

public void startDocument() throws SAXException {
  emit("<html>" + lineEnd);
}

public void endDocument() throws SAXException {
  emit("</html>" + lineEnd);
}

public void startElement(String namespaceURI,String sName,
  String qName,Attributes attrs) throws SAXException {
  String eName = sName; // element name
  if ("".equals(eName)) eName = qName; // namespaceAware = false
  stack.push(eName);
  value = new StringBuffer();
}

public void endElement(String namespaceURI,String sName,String qName)
  throws SAXException {
  String eName = (String)stack.pop();
  if (eName.equals("title") && stack.peek().equals("channel")) {
    emit("  <head>" + lineEnd);
    emit("  <title>" + value + "</title>" + lineEnd);
    emit("  </head>" + lineEnd);
    emit("  <body>" + lineEnd);
  } else if (eName.equals("title") &&
        stack.peek().equals("item")) {
    title = null == value ? "" : value.toString();
```

```
    } else if (eName.equals("link") &&
           stack.peek().equals("item")) {
      link = null == value ? "" : value.toString();
    } else if (eName.equals("description") &&
           stack.peek().equals("item")) {
      desc = null == value ? "" : value.toString();
    } else if (eName.equals("item")) {
      emit("  <p><a href=\"" + link + "\">" +
        title + "</a><br>" + lineEnd);
      emit("   " + desc + "</p>" + lineEnd);
    } else if (eName.equals("channel")) {
      emit("  </body>" + lineEnd);
    }
    value = null;
  }

  public void characters(char buf [], int offset, int len)
    throws SAXException {
    String s = new String(buf, offset, len);
    value.append(s);
  }

  private static void emit(String s) throws SAXException {
    try {
      out.write(s);
      out.flush();
    } catch (IOException e) {
      throw new SAXException("I/O error", e);
    }
  }
}
```

Compiling and running this program, we get the following results when we try to read XMLhack.com's RSS 1.0 feed (some lines have been wrapped for legibility):

```
C:\> java com.xml.RSSReader http://xmlhack.com/rss.php
<html>
  <head>
  <title>xmlhack</title>
  </head>
  <body>
  <p><a href="http://www.xmlhack.com/read.php?item=1511">Activity
around the Dublin Core</a><br>
   The Dublin Core Metadata Initiative (DCMI) has seen a
recent spate of activity,   Recent publications include The
Namespace Policy for the Dublin Core Metadata Initiative,

Expressing Simple Dublin Core in RDF/XML,

and Expressing Qualified Dublin Core in RDF/XML.</p>
  ...
```

An Intense and Bracing Introduction to C#

The most important thing to remember about C# is that although it's definitely *not* Java, it is a lot *like* Java. So much so that you can probably port a lot of your application logic with a simple search-and-replace operation. (I'll highlight those areas when we go over the relevant code examples.) There are a few syntactic differences between the languages; I'll mention a couple of the most interesting ones, but rather than go over them in detail, you should read about the language on your own. I recommend any or all of the following articles: Conversational C# for Java Programmers by Raffi Krikorian; C#: A language alternative or just J--? by Mark Johnson; and A Comparison of Microsoft's C# Programming Language to Sun Microsystems' Java Programming Language by Dare Obasanjo. There are also several books on C#, including Programming C# by Jesse Liberty and the forthcoming C# In A Nutshell by Peter Drayton & Ben Albahari. And of course, there is Microsoft's own invaluable .NET Framework Class Library.

Let's begin our introduction to C# by diving right into the code.

The first change really is a simple global search-and-replace. The C# equivalent of the `System.out` **class is Console, and the** `println()` **method's equivalent is** `WriteLine()`**.**

What we call packages in Java are called assemblies in C# -- an oversimplification, but it will do for purposes -- and they are brought into scope not with the import statement but with the using statement. Also, rather than declaring what package your class belongs to with a package statement, the assembly is declared through nested braces.

You don't have to be a rocket scientist to know that there is no `javax.xml.parsers` **assembly in C#. Microsoft has provided an assembly called** `System.Xml` **which contains all the XML classes you're likely to need.**

The last of our simple changes is one that may take some getting used to. Every Java developer knows that method names begin with lowercase letters, right? Well, Microsoft has taken a different route: building on its MFC naming tradition, all method names in C# begin with capital letters, including `Main()`.

Now let's see what those changes have done to our sample code so far.

```
using System;

public class com {
  public class xml {
  public class RSSReader {

    static private Writer out;
    static String lineEnd =  System.getProperty("line.separator");

    Stack stack = new Stack();
    StringBuffer value = null;
    String title = null;
    String link = null;
    String desc = null;
```

```
    public static void Main(String args[]) {

    // don't worry about the rest of this code yet
    // ...
    }
  }
  }
}
```

Compiling Your Code

Since we've made our trivial changes, we might as well let the compiler tell us what else we have to do. Yes, I know there's going to be a lot of error messages, but we have to start somewhere.

Of course you know the command line to compile and run the Java version of RSSReader (I'm running this on Windows, since that's where my C# project lives):

```
javac -g -classpath %CLASSPATH% com\xml\RSSReader.java
```

Here is the equivalent C# compilation command line:

```
csc /debug /r:System.Xml.dll /t:exe RSSReader.cs
```

Just as javac is the Java Compiler, `csc` **is the Cee-Sharp Compiler (get it?). I've listed the parameters in the same respective order in each compile line, so that you can see that the C# equivalent of** `-g` **is** `/debug` **and the Java** `-classpath` **parameter (which was not strictly necessary in this example, of course) is something like the** `/r` **switch. There's a final parameter on the C# compile line,** `/t:exe`**. In C#, you can run your code in the .NET runtime or you can compile an executable directly. In this case I'm compiling an executable, saving the wonderful world of .NET for future articles.**

If you run the compile line, you'll get a list of errors like the following.

```
Microsoft (R) Visual C# .NET Compiler version 7.00.9372.1
for Microsoft (R) .NET Framework version 1.0.3328
Copyright (C) Microsoft Corporation 2001. All rights reserved.

RSSReader1.cs(7,29): error CS1519:
        Invalid token 'out' in class, struct, or interface member
declaration
RSSReader1.cs(16,42): error CS1552:
        Array type specifier, [], must appear before parameter
name
...
```

Which should give you the impression that, although the languages are very similar, they're not identical. Maybe it's best not to try to compile it just yet. We're going to need to cover some minor syntax points first.

C# Minutiae

We need to make three very simple changes. First, out is a reserved word in C#, and, in fact, we should just delete that line and change all instances of `out.write()` **to** `Console.Write()`**.**

Second, in Java the brackets of an array declaration may come either after the type or after the instance name. In our case, the Java code is written as `String ar`**gs[]. In C#, the brackets must come after the type, thusly:** `String` **[]** `args`**. Another simple fix in two places.**

Finally, several of our Java methods have throws clauses. In C#, every exception is a runtime exception; there is no throws concept. We'll just delete all the throws clauses.

Compile again, and you'll see these errors:

```
Microsoft (R) Visual C# .NET Compiler version 7.00.9372.1
for Microsoft (R) .NET Framework version 1.0.3328
Copyright (C) Microsoft Corporation 2001. All rights reserved.

RSSReader1.cs(9,7): error CS0246:
       The type or namespace name 'Stack' could not be found
  (are you missing a using directive or an assembly reference?)
RSSReader1.cs(10,7): error CS0246:
       The type or namespace name 'StringBuffer' could not
  be found (are you missing a using directive or an
  assembly reference?)
RSSReader1.cs(48,22): error CS0246:
       The type or namespace name 'Attributes' could not be
  found (are you missing a using directive or an assembly
  reference?)
```

Which raises two more issues. Some of the Java classes that you've come to know and love don't exist in the C# world, though similar assemblies are available. And some do exist, but you've got to bring those assemblies into scope with the using statement. So our solution to these errors is twofold.

First, add the following lines at the top of your C# file.

```
using System;
using System.Collections;
using System.IO;
using System.Text;
```

Second, in addition to C# method names starting with a capital letter, there's another perversity: string starts with a *lower case* letter because it's actually a C# primitive. There are also some minor differences in the names of utility classes. For example, instead of `StringBuffer`**, C# has a** `StringBuilder` **class. Try to suppress your gag reflex, and do some global search-and-replaces.**

These changes will fix more compilation errors. And now we can learn about `XmlReader`**.**

XmlReader

`System.Xml` **contains many very useful classes. When you're talking about stream-based XML parsing, however, XmlReader is what you want..**

`XmlReader` **is most analogous to SAX, although it does not require implementing an interface as most SAX implementations do (as JAXP's does). Instead, you simply instantiate a concrete** `XmlReader` **of your choice -- there are several to choose from:** `XmlTextReader` **and** `XmlStreamReader` **are the most useful -- then call its Read() method and pick nodes off as they are returned to you.**

In algorithmic terms, we could say that SAX uses callbacks, while `XmlReader` **uses an event loop. An event loop is an infinite loop, during which certain events are received and dispatched to you. Internally, SAX may well have the same sort of event loop, but the callback methods hide that particular detail from you. A callback method is simply invoked when SAX's parser comes across an event.**

While that means that we don't need all those callback methods to satisfy an interface, we can still use them to dig through the nodes that we've read. In fact, we're just retrofitting a SAX-like API on top of `XmlReader`**. Here's the new version of the code; be sure and compare these to the Java version above.**

```
public static void Main(string [] args) {
  // create an instance of RSSReader
  RSSReader reader = new RSSReader();

  // parse the document from the parameter
  reader.Parse(args[0]);
}

// we have to write this method ourselves, since it's
// not provided by the API
public void Parse(string url) {
  try {
    XmlTextReader reader = new XmlTextReader(url);
    while (reader.Read()) {
      switch (reader.NodeType) {
      case XmlNodeType.Document:
        StartDocument();
        break;
      case XmlNodeType.Element:
        string namespaceURI = reader.NamespaceURI;
        string name = reader.Name;
        Hashtable attributes = new Hashtable();
        if (reader.HasAttributes) {
          for (int i = 0; i < reader.AttributeCount; i++) {
            reader.MoveToAttribute(i);
            attributes.Add(reader.Name,reader.Value);
          }
        }
        StartElement(namespaceURI,name,name,attributes);
        break;
      case XmlNodeType.EndElement:
        EndElement(reader.NamespaceURI,
```

```
                reader.Name, reader.Name);
        break;
      case XmlNodeType.Text:
        Characters(reader.Value,0,reader.Value.Length);
        break;
      // There are many other types of nodes, but
      // we are not interested in them
      }
    }
  } catch (XmlException e) {
    Console.WriteLine(e.Message);
  }
}

...

public void StartElement(string namespaceURI,string sName,
             string qName,Hashtable attrs) {
  string eName = sName; // element name
  if ("".Equals(eName))
    eName = qName; // namespaceAware = false
  stack.Push(eName);
  value = new StringBuilder();
}
```

You'll notice that `XmlReader` **has several interesting members, including** `NodeType`**,** `NamespaceURI`**, Name, and Value.** `NodeType` **can have several values depending on the node read from the XML document; we're only interested in Element,** `EndElement`**, and Text, though there are several other node types that you can use:** `CDATA`**, ProcessingInstruction, Comment,** `XmlDeclaration`**, Document,** `DocumentType`**, and** `EntityReference`**.**

As we receive one of the `NodeTypes` **we're interested in, we read the Name or Value and hand it off to our callback methods.**

One other point to note is that SAX has an Attributes class, for which the closest equivalent in C# is `XmlAttributeCollection`**. However, creating one of those is a more complex task than I really want to deal with right now, so instead, I've changed** `StartElement's` **last parameter to a** `Hashtable`**. I've done some fancy processing to populate it, but we're not interested in attributes for this example so I won't go into any more detail about it.**

Now that we've got our event loop and our "callback" methods in place, we can port the RSS-to-HTML conversion logic from Java to C#. This isn't exactly rocket science either, as the syntax of C# is basically the same as Java. Don't forget, though: all method names start with capital letters.

Here is the complete source listing for RSSReader.cs:

```
using System;
using System.Collections;
using System.IO;
using System.Text;
using System.Xml;
```

```
public class com {
  public class xml {
    public class RSSReader {
      static String LineEnd = "\n";

      Stack stack = new Stack();
      StringBuilder value = null;
      string title = null;
      string link = null;
      string desc = null;

      public static void Main(string [] args) {
        // create an instance of RSSReader
        RSSReader reader = new RSSReader();

        // parse the document from the parameter
        reader.Parse(args[0]);
      }

      // we have to write this method ourselves, since it's
      // not provided by the API
      public void Parse(string url) {
        try {
          XmlTextReader reader = new XmlTextReader(url);
          while (reader.Read()) {
            switch (reader.NodeType) {
            case XmlNodeType.Document:
              StartDocument();
              break;
            case XmlNodeType.Element:
              string namespaceURI = reader.NamespaceURI;
              string name = reader.Name;
              Hashtable attributes = new Hashtable();
              if (reader.HasAttributes) {
                for (int i = 0; i < reader.AttributeCount; i++) {
                  reader.MoveToAttribute(i);
                  attributes.Add(reader.Name,reader.Value);
                }
              }
              StartElement(namespaceURI,name,name,attributes);
              break;
            case XmlNodeType.EndElement:
              EndElement(reader.NamespaceURI,
                     reader.Name, reader.Name);
              break;
            case XmlNodeType.Text:
              Characters(reader.Value,0,reader.Value.Length);
              break;
            // There are many other types of nodes, but
            // we are not interested in them
            }
          }
        } catch (XmlException e) {
          Console.WriteLine(e.Message);
        }
      }
```

```
      public void StartDocument() {
        Emit("<html>" + LineEnd);
      }

      public void EndDocument() {
        Emit("</html>" + LineEnd);
      }

      public void StartElement(string namespaceURI,string sName,
                   string qName,Hashtable attrs) {
        string eName = sName; // element name
        if ("".Equals(eName)) eName = qName; // namespaceAware = false
        stack.Push(eName);
        value = new StringBuilder();
      }

      public void EndElement(string namespaceURI,string sName,
                   string qName) {
        string eName = (string)stack.Pop();
        if (eName.Equals("title") &&
            stack.Peek().Equals("channel")) {
          Emit("  <head>" + LineEnd);
          Emit("  <title>" + value + "</title>" + LineEnd);
          Emit("  </head>" + LineEnd);
          Emit("  <body>" + LineEnd);
        } else if (eName.Equals("title") &&
                   stack.Peek().Equals("item")) {
          title = null == value ? "" : value.ToString();
        } else if (eName.Equals("link") &&
                   stack.Peek().Equals("item")) {
          link = null == value ? "" : value.ToString();
        } else if (eName.Equals("description") &&
                   stack.Peek().Equals("item")) {
          desc = null == value ? "" : value.ToString();
        } else if (eName.Equals("item")) {
          Emit("  <p><a href=\"" + link + "\">" +
                title + "</a><br>" + LineEnd);
          Emit("   " + desc + "</p>" + LineEnd);
        } else if (eName.Equals("channel")) {
          Emit("  </body>" + LineEnd);
        }
        value = null;
      }

      public void Characters(string buf, int offset, int len) {
        value.Append(buf);
      }

      private static void Emit(string s) {
        Console.Write(s);
      }
    }
  }
}
```

Try it, and you'll see that it compiles without a problem. Run it like so (output wrapped for legibility):

```
C:\> RSSReader http://xmlhack.com/rss.php
  <head>
  <title>xmlhack</title>
  </head>
  <body>
  <p><a href="http://www.xmlhack.com/read.php?item=1511">Activity
around the Dublin Core</a><br>
   The Dublin Core Metadata Initiative (DCMI) has seen a recent
spate of activity,
       Recent publications include The Namespace Policy for the
Dublin Core Metadata
       Initiative, Expressing Simple Dublin Core in RDF/XML, and
Expressing Qualified
       Dublin Core in RDF/XML.</p>
...
```

This should look very familiar, as it's exactly the same output that our Java program produced. You might have seen a completely different result, however, like this:

```
C:\> RSSReader http://xmlhack.com/rss.php

Unhandled Exception: System.Security.SecurityException:
    Request for the permission of type
        System.Net.WebPermission, System, Version=1.0.3300.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089 failed.
   at System.Security.CodeAccessSecurityEngine.CheckHelper(
    PermissionSet grantedSet,
    PermissionSet deniedSet, CodeAccess
    Permission demand, PermissionToken permToken)
   at System.Security.CodeAccessSecurityEngine.Check(
    PermissionToken permToken,
    CodeAccessPermission demand, StackCrawlMark& stackMark,
    Int32 checkFrames, Int32 unrestrictedOverride)
   at System.Security.CodeAccessSecurityEngine.Check(
    CodeAccessPermission cap, StackCrawlMark& stackMark)
   at System.Security.CodeAccessPermission.Demand()
   at System.Net.HttpRequestCreator.Create(Uri Uri)
   at System.Net.WebRequest.Create(Uri requestUri,
    Boolean useUriBase)
   at System.Net.WebRequest.Create(Uri requestUri)
   at System.Xml.XmlDownloadManager.GetNonFileStream(Uri uri,
    ICredentials credentials)
   at System.Xml.XmlDownloadManager.GetStream(Uri uri,
    ICredentials credentials)
   at System.Xml.XmlUrlResolver.GetEntity(Uri absoluteUri,
    String role, Type ofObjectToReturn)
   at System.Xml.XmlTextReader.CreateScanner()
   at System.Xml.XmlTextReader.Init()
   at System.Xml.XmlTextReader.Read()
   at RSSReader.Parse(String url) in U:\thing\RSSReader.cs:line 31
   at RSSReader.Main(String[] args) in U:\thing\RSSReader.cs:line 23
```

The state of the failed permission was:

```
<IPermission class="System.Net.WebPermission, System,
    Version=1.0.3300.0, Culture=neutral,
    PublicKeyToken=b77a5c561934e089" version="1">
   <ConnectAccess>
    <URI uri="http://xmlhack\.com/rss\.php"/>
   </ConnectAccess>
</IPermission>
```

This happens because C# will not load an assembly on a network drive. Just move your RSSReader.exe executable onto a local drive and try again.

Conclusions

What can we learn from `XmlReader`? First of all, unlike Java's XML libraries, all of S`ystem.Xml` is provided by Microsoft. This means that, among other things, there is a consistent interface and a consistent set of tools for all your XML needs. No need to shop around for parsers and SAX implementations.

That can also be considered a drawback. Since Java has multiple implementations, you're free to use the one that fits best. And with the advent of JAXP, you can drop in the different implementations without changing your code at all. Doing that in C# is, well, impossible; you're stuck with the one, true Microsoft way.

As far as the `XmlReader` event model, it doesn't seem that SAX has much to learn at all. You'll remember that we actually had to write some additional code when we ported the program to `XmlReader` because SAX provided the event loop for us; with SAX, all we have to do is write the callbacks.

On the other hand, `System.Xml` *does* provide some nifty classes like `XmlAttribute` (which we didn't really discuss here) and `XmlNodeType`, which give you very convenient, standard ways to look at attributes and nodes, instead of having to deal with strings and Hashtables and such. And all these classes are used throughout C#'s XML facilities.

If you don't want to write either an event loop or callbacks, the read-only, forward-only, stream-based model might not be for you; you might prefer a whole-document model (like, say, DOM). In that case, XmlReader will not appeal to you any more than SAX does. There is another set of tools in C#, starting with XmlDocument, which we'll discuss in the next article, which gives you all the power of a document stored in memory, plus the added convenience of building on what you've already learned.

~~~ End of Article ~~~