XOM

By Ken Cochrane

XML Object Model (XOM) is a tree based Java API for XML, written by Elliotte Rusty Harold. Taking the best ideas from SAX and DOM, Elliotte created an API that allows for the processing and creating of XML. Elliotte's major goals for XOM was to make it simple, so that it is easy to use, fast, small enough and has no gotchas. From what I have seen with the 1.0d12 release he has accomplished that and more.

Before we get into how XOM works let me first discuss what a tree based parser is, and why it is, in some ways, better than the other types of parsers. In Listing 1, you will see a very basic XML representation of an address book. The address_book element is at the top of the XML document, so we call it the root element. Inside the address_book element are 2 "entry" elements, 1 for Ken and another for Emily. The "entry" element is a child of address_book. The entry element has 3 child elements of its own first_name, last_name, and email. For more details on the approved structure for this XML document see Listing 2 which lists its DTD.

If we were to take the address book XML document and represent it as a tree it would look like Figure 1. Notice how address_book is at the top of the tree and it has 2 entry elements below it, and below each entry element it has a first_name last_name and email element. Notice how the diagram looks like an upside-down tree with the address book being the root of the tree.

```
<address_book>
        <entry>
                <first_name>Ken</first_name>
                <last_name>Cochrane</last_name>
                <email>ken@fakeURL.no</email>
        </entry>
        <entry>
                <first_name>Emily</first_name>
                <last_name>Cochrane</last_name>
                <email>Emily@fakeURL.no</email>
        </entry>
</address_book>
```

Listing 1. address_book.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT email (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT entry (first_name,last_name,email)>
```

```
<!ELEMENT address_book (entry+)>
<!ATTLIST address_book xmlns CDATA #REQUIRED>
```

Listing 2. address book DTD

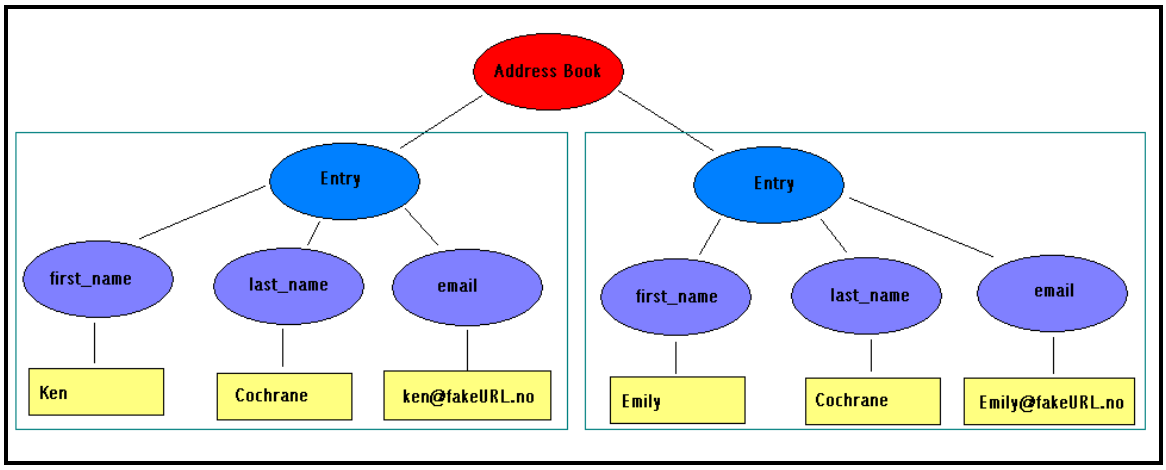

Figure 1 - Address Book Node Tree

Now that we have a visual of what our XML document looks like let's see how we can use it when parsing the document with XOM.

To get started, I have created a very simple java class which you can see in Figure X. It has a main method, which will get called when you execute this from the command line. It also has a method called parse which takes in a parameter of type string called file. The parse method is currently set as a void, because right now we are just going to open up the XML document, parse it and display the results to the user via system.out statements.

This class will get called from the command line and we will pass the name of the XML file as the first parameter. Inside of the main method we take the first parameter and pass it to the parse method.

Inside of the parse method, the first thing we need to do is check and make sure the file exists. We do this with a simple try and catch block. We try to open up a FileInputStream with the filename and if the file isn't there the exception will be caught by the catch block and display a message to the user letting them know what happened and then exit.

In Listing 3 we are going to create an XOM builder object, which will help us, parse the XML document and then we will find the root element of the document and with that find all of the child elements it has.

```
        FileInputStream xmlFile = new FileInputStream(file);
```

```
Builder builder = new Builder();

Document doc = builder.build(xmlFile);

Element root = doc.getRootElement();

Elements entries = root.getChildElements();
```

Listing 3. Sample code

The `Builder` class is responsible for creating XOM `Document` objects from a URL, file, or input stream by reading an XML document. A SAX parser is used to read the document and report any well-formedness errors. We use the builder class to parse our XML file and create an XOM document object.

Using the document objects getRootElement() method we get the root element of the document, which in this case is the address_book element. With the root element we can then get all of its children elements by calling the getChildElements() method which returns an array of elements.

With the array of elements we can use a simple for loop to cycle through each of the element and do what we want. For this example I have decided to just print them to the screen. Below you should see the for loop code in Listing 4, and below that the output for our simple parse method is in Listing 5.

```
//print out the entries
 for (int x = 0; x < entries.size(); x++) {
    Element element = entries.get(x);
    System.out.println(element.getLocalName());
    System.out.println(element.getValue());
 }
```

Listing 4. For Loop to cycle through entries

```
entry

        Ken
        Cochrane
        ken@fakeURL.no

entry

        Emily
        Cochrane
        emily@fakeURL.no
```

Listing 5. Output for example 1

See ABook.java for the complete set of code.

The previous piece of code is good for displaying what is in the XML document, but it wasn't good for much else since it just printed everything in one big blob. In order to do something with the data that we get from the XML file we will need to parse the document with a little more detail. Since we know the structure of the XML document it makes parsing it that much easier. In the code above we just printed every element that we found in the XML file. If we use Elements getFirstChildElement method we can grab the child elements that we want and do what ever we please with them. I have rewritten the for loop code from above to search for the elements first_name, last_name and email, and then, when found, print them to the screen. The code snippet is located below. We get the element the same way we did above, but we added a few things. First off we check to make sure that the element has children. If it doesn't there isn't any reason to continue. Then we use the getFirstChildElement, which we discussed before, to get the child elements we want. Once we have the child elements, we check and make sure they aren't null and we print them out to the screen using System.out statements. The new output is shown below.

```
for (int x = 0; x < entries.size(); x++) {
    Element element = entries.get(x);

    if (element.hasChildren()) {
        System.out.println("---");
        Element firstName = element.getFirstChildElement("first_name");

        if (firstName != null) {
            System.out.print("First Name = ");
            System.out.println(firstName.getValue());
        }

        Element lastName = element.getFirstChildElement("last_name");

        if (lastName != null) {
            System.out.print("Last Name = ");
            System.out.println(lastName.getValue());
        }

        Element email = element.getFirstChildElement("email");

        if (email != null) {
            System.out.print("email = ");
            System.out.println(email.getValue());
        }
    }
}
```

Listing 6.

```
---
First Name = Ken
Last Name = Cochrane
email = ken@fakeURL.no
---
First Name = Emily
Last Name = Cochrane
email = emily@fakeURL.no
```

Listing 7.

See Abook2.java for the full working code.

Now that we can successfully get the data from all of the nodes in the XML file we can start to have some fun. Let's parse the document like before but instead of just outputting the data to the screen let's change some of the data then print it to the screen. This should show you how easy it is to take one XML file, parse it, change some of the data and then output the new update XML file.

In the code snippet below you will see that we have changed the email code a bit. We want to take all of the email addresses that are in the XML document and change them to the new format. The new format is "first_name@NewPlace.yes" where first_name is the value that is in the XML file for the element first_name. We print out the values like before, but after that we remove the data in the element by removing the child, and we add the new value to the element by calling the appendChild method. To see if this works correctly I have added a System.out statement that prints out the current document as an XML file. See the new output below.

```
        Element email = element.getFirstChildElement("email");

        if (email != null) {
            System.out.print("email = ");
            System.out.println(email.getValue());
            email.removeChild(0);
            email.appendChild(firstName.getValue() + "@NewPlace.yes");
        }
           .
           .
        System.out.println(doc.toXML());
```

Listing 8.

```
---
First Name = Ken
Last Name = Cochrane
email = ken@fakeURL.no
```

```
---
First Name = Emily
Last Name = Cochrane
email = emily@fakeURL.no
---
<?xml version="1.0"?>
<address_book>
	<entry>
		<first_name>Ken</first_name>
		<last_name>Cochrane</last_name>
		<email>Ken@NewPlace.yes</email>
	</entry>
	<entry>
		<first_name>Emily</first_name>
		<last_name>Cochrane</last_name>
		<email>Emily@NewPlace.yes</email>
	</entry>
</address_book>
```

Listing 9. Output for example 3

See Abook3.java for the full working code.

Changing the information on the fly like we did in the last example is OK for small changes, but it wouldn't cut it if we needed to make many changes. We need a process to take the data from the XML file and store it in a way that makes it easy to manipulate. Then, we do what we want with it, like printing it out to the screen. I accomplish this by creating two very simple Java beans. The first Java bean is called Person. It holds a first and last name, and an email. The person Java bean matches the entry element inside of the address book XML file. The other Java bean that I created is called AddressBook. It holds an array of Person beans. The AddressBook bean can hold 0 or many Person objects just like the AddressBook.xml file can hold 0 or many entry elements. With both of the Java beans I can hold all of the information from the XML file, natively inside of Java which will make dealing with all the data that much easier. For more details on the Person and AddressBook Java beans see Person.java and AddressBook.java.

In order to accommodate the new Java Beans, I have changed the parse method from a void, which returns nothing to return an AddressBook object. Along with this change was the removal of all of the system.out statements, which where replaced with code that creates a Person object and fills it up with the information for the XML file. When, the parse is complete it returns the filled up AddressBook object. The code below shows all of the changes to the parse method.

```
public static AddressBook parse(String file) {
	AddressBook aBook = new AddressBook();
	Person person = new Person();
	try {
				.
				.  //some code removed for space reasons
```

```
                    .
        for (int x = 0; x < entries.size(); x++) {
            Element element = entries.get(x);

            if (element.hasChildren()) {
                person = new Person();

                Element firstName = element.getFirstChildElement("first_name");

                if (firstName != null) {
                    person.setFirstName(firstName.getValue());
                }

                Element lastName = element.getFirstChildElement("last_name");

                if (lastName != null) {
                    person.setLastName(lastName.getValue());
                }

                Element email = element.getFirstChildElement("email");

                if (email != null) {
                    person.setEmail(email.getValue());
                }

                if (person != null) {
                    aBook.addElement(person);
                }
            }
        }
        .
        .  //catch statements removed for space reasons
        .
        finally {
            return aBook;
        }
    }
```

Listing 10.

Some of the code was removed to save space see ABook4.java for full source code.

Since we removed the system.out statements from parse we need a new way to print out the results to the screen, so a new method called printAddressBook was added. PrintAddressBook does just that, it cycles through the AddressBook bean and prints out all of the information to the screen. The code is shown below.

```
private static void printAddressBook(AddressBook aBook) {
        Person person = new Person();
        if (aBook != null && aBook.size() > 0) {
            for (int x = 0; x < aBook.size(); x++) {
```

```
            person = aBook.elementAt(x);
            if (person != null) {
               System.out.println("----");
               System.out.println("First Name = " + person.getFirstName());
               System.out.println("Last Name = " + person.getLastName());
               System.out.println("Email = " + person.getEmail());
            }
         }
      }
   }
```

Listing 11.

In order to accommodate these two new methods, the main method had to change a little. We need to capture the AddressBook object that is now getting returned from the parse method, and we need to pass that object to the printAddressBook method to get the results printed. The changes are shown below. Some of the code is removed to save space. See ABook4.java for the full source code.

```
            AddressBook aBook = new AddressBook();
            aBook = parse(file);
             if (aBook != null && aBook.size() > 0) {
                printAddressBook(aBook);
            }
```

Listing 12.

The output for all of the new changes is shown below. Notice that the output looks the same as before. That is the reason we wanted to change the way the program was written, to make our job easier, but at the same time, not change the output. We accomplished what we were trying to do.

```
----
First Name = Ken
Last Name = Cochrane
Email = ken@fakeURL.no
----
First Name = Emily
Last Name = Cochrane
Email = emily@fakeURL.no
```

Listing 13.

Now that we have the new data structures that makes our job easier let's see how they work. Let's try and accomplish what we did earlier with the old code. We will change the email addresses from the current one to the new "first_name@NewPlace.yes" email addresses.

With the new Java beans it is a lot easier to change the email address since we don't have to play with any of the XOM methods. All that is needed in order to change the email addresses is a new method that will cycle through the AddressBook bean and change the email value for all of the Person objects. This is accomplished with the new method called changeAddressBook, which is shown below in Listing 14.

```
private static void changeAddressBook(AddressBook aBook) {
      Person person = new Person();
      if (aBook != null && aBook.size() > 0) {
         for (int x = 0; x < aBook.size(); x++) {
            person = aBook.elementAt(x);
            if (person != null) {
               person.setEmail(person.getFirstName() + "@NewPlace.yes");
            }
         }
      }
   }
```

Listing 14.

Once the AddressBook object is changed all we need to do is call the printAddressBook method again. Now, wasn't that a lot easier to deal with then before? The output is shown below. The output shows the original AddressBook data followed by the new changed data.

```
--Before--
----
First Name = Ken
Last Name = Cochrane
Email = ken@fakeURL.no
----
First Name = Emily
Last Name = Cochrane
Email = emily@fakeURL.no
--After--
----
First Name = Ken
Last Name = Cochrane
Email = Ken@NewPlace.yes
----
First Name = Emily
Last Name = Cochrane
Email = Emily@NewPlace.yes
```

Listing 15.

Printing to the screen is good for development purposes, but for real applications you will most likely need to save the AddressBook data back to a file. This is accomplished with a new method called saveXMLToFile. SaveXMLToFile takes in an XOM Document object and a filename and creates an XML file with the name passed into it. The code is shown below. In the code, the most important stuff to note is the Serializer. The Serializer

is basically what controls the look of the XML when it gets printed to the file. Without the Serializer, the XML would all be on one line and it would be hard to read. The Serializer will add indents and line feeds to make the output readable. In this example we use a FileOutputStream with the Serializer but you can use other outputs if you like. Later on you will see that I use System.out with the Serializer to make printing XML to the screen look much nicer.

```
public static void saveXMLToFile(Document doc, String fileName) {
      try {
         System.out.println("Saving to File " + fileName);
         File outfile = new File(fileName);
         FileOutputStream fos = new FileOutputStream(outfile.getName());
         Serializer output = new Serializer(fos, "ISO-8859-1");
         output.setIndent(2);
         output.write(doc);
      } catch (FileNotFoundException fnfe) {
         System.out.println("File Not Found");
         fnfe.printStackTrace();
         System.exit(-1);
      } catch (UnsupportedEncodingException uee) {
         System.out.println("unsupported Exception");
         uee.printStackTrace();
         System.exit(-1);
      } catch (IOException ioe) {
         System.out.println("IO Exception");
         ioe.printStackTrace();
         System.exit(-1);
      }
}
```

Listing 16.

You might have noticed that the saveXMLToFile method accepts a XOM Document object, yet all of our changes to the XML file are currently getting held in the AddressBook object. You might be wondering how we get the AddressBook printed out to a file. This is accomplished with a new method that is called createXML. The createXML method takes in an AddressBook object and returns an XOM Document object. The code for createXML is shown below in Listing 17.

```
   public static Document createXML(AddressBook aBook) {
      Person person = new Person();
      //create the elements
      Element addressBook = new Element("address_book");
      Element entry = new Element("entry");
      Element firstName = new Element("first_name");
      Element lastName = new Element("last_name");
      Element email = new Element("email");
      //create the tree structure
      Document doc = new Document(addressBook); //root element
```

```
        if (aBook != null && aBook.size() > 0) {
            for (int x = 0; x < aBook.size(); x++) {
                person = aBook.elementAt(x);
                if (person != null) {
                    entry = new Element("entry");
                    firstName = new Element("first_name");
                    lastName = new Element("last_name");
                    email = new Element("email");

                    addressBook.appendChild(entry);
                    entry.appendChild(firstName);
                    firstName.appendChild(person.getFirstName());
                    entry.appendChild(lastName);
                    lastName.appendChild(person.getLastName());
                    entry.appendChild(email);
                    email.appendChild(person.getEmail());
                }
            }
        }
        return doc;
    }
```

Listing 17.

The first thing that you need to do in order to create an XOM Document object is to create all of your elements. Once all of your elements are created, you need to create the document structure by appending the child elements to their parents. Once the document structure is created, you can append the data from the AddressBook object into the appropriate XOM elements. When all of the data from the AddressBook object has been added to the Document object, then you are finished. Now that you have the completed Document object all filled up with the data from the AddressBook object you can send it to the saveXMLToFile method to save the data to a file. I have shown the output below. In the output you will notice two sets of XML code. The first set isn't Serialized and because of this the output is hard to read and it spans the entire width of the page. Whereas the second snippet of XML code, which is Serialized, is well formatted and thus easier to read. Now you can see the difference between Serialized an unSerialized.

```
--Before--
----
First Name = Ken
Last Name = Cochrane
Email = ken@fakeURL.no
----
First Name = Emily
Last Name = Cochrane
Email = emily@fakeURL.no
--After--
----
First Name = Ken
Last Name = Cochrane
```

```
Email = Ken@NewPlace.yes
----
First Name = Emily
Last Name = Cochrane
Email = Emily@NewPlace.yes
--XML--
<?xml version="1.0"?>
<address_book><entry><first_name>Ken</first_name><last_name>Cochrane
</last_name><email>Ken@NewPlace.yes</email></entry><entry><first_name>
Emily</first_name><last_name>Cochrane</last_name><email>Emily@NewPlace.yes
</email></entry></address_book>

Saving to File output.xml

C:\ken\classes\JavaAndXML\hw6\build>type output.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<address_book>
  <entry>
    <first_name>Ken</first_name>
    <last_name>Cochrane</last_name>
    <email>Ken@NewPlace.yes</email>
  </entry>
  <entry>
    <first_name>Emily</first_name>
    <last_name>Cochrane</last_name>
    <email>Emily@NewPlace.yes</email>
  </entry>
</address_book>
```

Listing 18.

See ABook6.xml for full source code.

The XML file that we have been parsing so far has been very basic. It didn't have Namespace, DTD, or XML Schema information in it. Since these three items are very common in XML files let's see how adding these elements to our XML file affects our parsing technique. We will start off by adding DTD and Namespace information to our original XML file; the resulting XML file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE address_book SYSTEM "address_book.dtd">
<address_book xmlns="http://www.popcornmonsters.com/">
        <entry>
                <first_name>Ken</first_name>
                <last_name>Cochrane</last_name>
                <email>ken@fakeURL.no</email>
        </entry>
        <entry>
                <first_name>Emily</first_name>
                <last_name>Cochrane</last_name>
                <email>emily@fakeURL.no</email>
```

```
        </entry>
</address_book>
```

Listing 19.

Since the XML file has changed, we will need to change the parse method. The new parse method is shown below. The first thing you should notice is that we have change the method signature to include the namespace for the document. We will pass in the namespace so that we can always change it in the future without the risk of having to change to much code. Most of the code doesn't change except for the getChildElements and getFirstChildElments methods. We are now passing into these two methods the namespace as the second parameter. If we didn't pass in the namespace to these methods then XOM wouldn't be able to find what we are looking for in the document. Adding the DTD to the XML file doesn't affect the parse method at all.

```
public static AddressBook parse(String file, String nameSpace) {
      AddressBook aBook = new AddressBook();
      Person person = new Person();
      try {
         FileInputStream xmlFile = new FileInputStream(file);

         Builder builder = new Builder();
         Document doc = builder.build(xmlFile);
         Element root = doc.getRootElement();

         // getChildElements
         Elements entries = root.getChildElements("entry", nameSpace);

         for (int x = 0; x < entries.size(); x++) {
            Element element = entries.get(x);
            if (element.hasChildren()) {
               person = new Person();

     Element firstName = element.getFirstChildElement("first_name", nameSpace);

               if (firstName != null) {
                  person.setFirstName(firstName.getValue());
               }

     Element lastName = element.getFirstChildElement("last_name", nameSpace);

               if (lastName != null) {
                  person.setLastName(lastName.getValue());
               }

     Element email = element.getFirstChildElement("email", nameSpace);
```

```
                if (email != null) {
                    person.setEmail(email.getValue());
                }

                if (person != null) {
                    aBook.addElement(person);
                }
            }
        }
        .
        .  //catches removed to save space
        .
         finally {
            return aBook;
        }
    }
```

Listing 20.

Changing the XML input file also means that we need to update the CreateXML method so that our output matches our new input. The new code is shown below. The major change to the code is adding namespace as the second parameter when we create a new Element. Adding the namespace when we create the new element XOM will take care of all of the dirty work behind the scenes for us. The most important part is to make sure to add the namespace to all affected Elements, because if you forget just one you won't get the output that you desire. In order to add the DTD information to the XML file you will need to create a new DocType object and then insert the DocType as a child to the document object. See code below for more details.

```
public static Document createXML(AddressBook aBook, String nameSpace) {
        Person person = new Person();
        //create the elements
        Element addressBook = new Element("address_book", nameSpace);
        Element entry = new Element("entry", nameSpace);
        Element firstName = new Element("first_name", nameSpace);
        Element lastName = new Element("last_name", nameSpace);
        Element email = new Element("email", nameSpace);
        //create the tree structure
        Document doc = new Document(addressBook); //root element

        //add the dtd
        DocType docType = new DocType("address_book", "address_book.dtd");
        doc.insertChild(0, docType);

        if (aBook != null && aBook.size() > 0) {
            for (int x = 0; x < aBook.size(); x++) {
                person = aBook.elementAt(x);
```

```
            if (person != null) {
                entry = new Element("entry", nameSpace);
                firstName = new Element("first_name", nameSpace);
                lastName = new Element("last_name", nameSpace);
                email = new Element("email", nameSpace);

                addressBook.appendChild(entry);
                entry.appendChild(firstName);
                firstName.appendChild(person.getFirstName());
                entry.appendChild(lastName);
                lastName.appendChild(person.getLastName());
                entry.appendChild(email);
                email.appendChild(person.getEmail());
            }
        }
    }
    return doc;
}
```

Listing 21.

The output after we made all of the changes is shown in Listing 22. As you can see, the output XML file now includes the nameSpace and DTD information.

```
--Before--
----
First Name = Ken
Last Name = Cochrane
Email = ken@fakeURL.no
----
First Name = Emily
Last Name = Cochrane
Email = emily@fakeURL.no
--After--
----
First Name = Ken
Last Name = Cochrane
Email = Ken@NewPlace.yes
----
First Name = Emily
Last Name = Cochrane
Email = Emily@NewPlace.yes
--XML--
<?xml version="1.0"?>
<!DOCTYPE address_book SYSTEM "address_book.dtd">
<address_book xmlns="http://www.popcornmonsters.com/"><entry><first_name>Ken
</first_name><last_name>Cochrane</last_name><email>Ken@NewPlace.yes</email>
</entry><entry><first_name>Emily</first_name><last_name>Cochrane</last_name>
<email>Emily@NewPlace.yes</email></entry></address_book>
```

```
Saving to File output.xml

C:\ken\classes\JavaAndXML\hw6\build>type output.xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE address_book SYSTEM "address_book.dtd">
<address_book xmlns="http://www.popcornmonsters.com/">
  <entry>
    <first_name>Ken</first_name>
    <last_name>Cochrane</last_name>
    <email>Ken@NewPlace.yes</email>
  </entry>
  <entry>
    <first_name>Emily</first_name>
    <last_name>Cochrane</last_name>
    <email>Emily@NewPlace.yes</email>
  </entry>
</address_book>
```

Listing 22.

Now that we have DTD and namespaces added to the XML file, let's add an XML schema. The XML schema that we will be using for this example is located below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.popcornmonsters.com"
    xmlns="http://www.popcornmonsters.com"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:element name="email" type="xs:string"/>
        <xs:element name="first_name" type="xs:string"/>
        <xs:element name="last_name" type="xs:string"/>
    <xs:element name="entry">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="first_name" minOccurs="0"/>
                                <xs:element ref="last_name" minOccurs="0"/>
                                <xs:element ref="email" minOccurs="0"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
    <xs:element name="address_book">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="entry" minOccurs="0"/>
                        </xs:sequence>
```

```
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Listing 23.

In order to use this XML schema we need a way to link the XML document to this schema. After we add the information to link the schema and the XML document, the AddressBook.xml file looks like this.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE address_book SYSTEM "address_book.dtd">
<address_book xmlns="http://www.popcornmonsters.com/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.popcornmonsters.com
address_book.xsd">
        <entry>
                <first_name>Ken</first_name>
                <last_name>Cochrane</last_name>
                <email>ken@fakeURL.no</email>
        </entry>
        <entry>
                <first_name>Emily</first_name>
                <last_name>Cochrane</last_name>
                <email>emily@fakeURL.no</email>
        </entry>
</address_book>
```

Listing 24.

This change doesn't affect the Parse method at all. The only code that we need to change is in the CreateXML method so that the XML file that gets created matches the new format. All that we need to do in order to change the XML file is to add an attribute to the addressbook object. The attribute method takes a key value pair and then appends it into the address_book element in the XML file. The key that we want to add is "schemaLocation" and the value is "http://www.popcornmonsters.com/address_book.xsd". Once we create the Attribute information, we need to set the namespace for it by using the setNamespace method. Now that the attribute object is all created we can add the attribute to the address_book element. An example is shown below. See ABook8.java for source code.

```
Attribute attrib = new Attribute("schemaLocation",
"http://www.popcornmonsters.com/address_book.xsd");

attrib.setNamespace("xsi", "http://www.w3.org/2001/XMLSchema-instance");
addressBook.addAttribute(attrib);
```

Listing 25.

With the new changes to the CreateXML method the output now looks like this. Notice the new attribute for the address_book element.

```
//Output edited to save space.

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE address_book SYSTEM "address_book.dtd">
<address_book
xsi:schemaLocation="http://www.popcornmonsters.com/address_book.xsd"
xmlns="http://www.popcornmonsters.com/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <entry>
    <first_name>Ken</first_name>
    <last_name>Cochrane</last_name>
    <email>Ken@NewPlace.yes</email>
  </entry>
  <entry>
    <first_name>Emily</first_name>
    <last_name>Cochrane</last_name>
    <email>Emily@NewPlace.yes</email>
  </entry>
</address_book>
```

Listing 26.

To make this class a little more useful I have added the following methods prettyScreenPrint, addEntry and RemoveEntry. I won't go into much details about each one other then to just give a brief description about what each does, and show the source code.

The prettyScreenPrint method is fairly simple. It allows the user to print the completed XML file to the screen in an easy to read format. The code is located below.

```
    public static void prettyScreenPrint(Document doc) {
        try {
            Serializer serializer = new Serializer(System.out, "ISO-8859-1");
            serializer.setIndent(4);
            serializer.setMaxLength(64);
            serializer.write(doc);
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
```

Listing 27.

The addEntry and removeEntry methods are just that. They allow for entries to be added or removed from the address book. These two methods don't have anything to do with

XOM, but they make the application more well rounded. The source code is shown below in Listing 28.

```
public static void addEntry(AddressBook aBook, String fName, String lName, String
email) {
        Person person = new Person();
        person.setFirstName(fName);
        person.setLastName(lName);
        person.setEmail(email);

        aBook.addElement(person);
    }

public static void removeEntry(AddressBook aBook, String fName, String lName) {
        Person person = new Person();
        for (int x = 0; x < aBook.size(); x++) {
            person = aBook.elementAt(x);
            if (person != null) {
                if (person.getFirstName().equalsIgnoreCase(fName) &&
                        person.getLastName().equalsIgnoreCase(lName)) {
                    aBook.removeElement(x);
                }
            }
        }
    }
```

Listing 28.

As of now we have always assumed that the XML file we were parsing was valid, which isn't always the case. It is good practice to validate the XML file before you attempt to parse it. Fortunately XOM has a built in function that will validate the XML file for you, and if there is an error, it will throw a ValidityException and it won't go any further. All that you need to do in order to turn on validating is pass a true value to the Build object when you are creating it. See the code in Listing 29 below for an example.

```
Builder builder = new Builder(true);
```

Listing 29.

Sometimes, when you are creating an XML file, you need to add a comment to make the XML file a little more descriptive. XOM makes this easy as well. All you need to do is create a new Comment object with the comment that you want to put into the XML file and then insert it as a child to what ever element you want it to appear in. The code is shown below in Listing 30.

```
Comment detailsComment =
   new Comment("Address Book for Ken Cochrane Last updated on " +
                        new java.util.Date());
addressBook.insertChild(0, detailsComment);
```

Listing 30.

That concludes my introduction to XOM. I hope you have seen how easy it is to parse an XML document with Java. This was never suppose to be a document that explained everything about XOM, it was suppose to show you how to use XOM to complete the common tasks associated with parsing and creating XML with Java using XOM. Now you should have enough knowledge to be dangerous, go out and have some fun. Happy XOMing!

Requirements

- SAX2 compliant parser installed
- Java 1.2 or greater

Resources

- XOM Home Page - http://www.cafeconleche.org/XOM/ or http://www.xom.nu/
- XOM Article on XML.com - http://www.xml.com/pub/a/2002/11/27/xom.html?page=1
- Linux Magazine article - Java XOM - *Rogers Cadenhead* – March 2003 - (http://www.linux-mag.com/2003-03/toc.html)

Installation

- Download the XOM jar file from the XOM home page and add it to your class path

Running examples

All of my code was written with XOM version 1.0d10. As of 05/10/2003 1.0d12 was released. I used Xerces 2.4, which comes with XOM, for my SAX2 parser. $JARS is a system variable that is pointing to my JARS directory.

- ABook.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook AddressBook.xml
- ABook2.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook2 AddressBook.xml
- ABook3.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook3 AddressBook.xml

- ABook4.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook4 AddressBook.xml
- ABook5.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook5 AddressBook.xml
- ABook6.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook6 AddressBook.xml
- ABook7.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook7 AddressBook2.xml
- ABook8.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook8 AddressBook3.xml
- ABook9.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook9 AddressBook3.xml
- ABook10.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook10 AddressBook3.xml
- ABook11.java
  - java -classpath $JARS\XOM.jar; $JARS\Xerces.zip;$JARS\xercesImpl.jar;$JARS\jars\xmlParserAPIs.jar; com.popcornmonsters.hw6.ABook11 AddressBook4.xml

Coming soon in XOM

- At least partial XPath support. This will probably be limited to location paths and compound location paths, which is what people really need inside Java anyway.
- Vector functions that operate on every node in a list.
- Filters for builders and writers.
- XSLT support. (He has begun to work on this, but it's not actually working yet, and it may not be in time for 1.0.)

Source code

- The full source code that is mentioned throughout this document is located in the src directory in the zip file that accompanied this document. I will

try and post everything on my website when I get a chance, and when I do I will list the URL here.

Mailing list

- If you would like to say up to date with the daily development of XOM sign up for the XOM mailing list at this URL http://lists.ibiblio.org/mailman/listinfo/xom-interest

Javadocs

- The Javadocs for these code examples as well as for XOM is located in the docs directory in the zip file that accompanied this document