Service-Centric Cloud Technologies - Practical Study #7

XML Schema Tutorial

Part I *of this exercise focuses on the familiarisation with NetBeans tools for building and verifying the correctness of XML Schemas.*

Part II *uses the Java Architecture for XML binding (JAXB) to generate Java classes from a pre-defined XML Schema to populate or read a corresponding XML document.*

Part III *demonstrates how to work with a 'collection of items' in XML.*

Part I Working with XML Schemas in NetBeans

1. Download the source code folder "**JAXBIntro.zip**" from the module site and unzip it into your local drive. Create a new **JAXBIntro** project *with Existing Sources*.
2. Examine the "Books.xsd" file under (source Packages → <default package>) and make sure you understand the structure of the file.
3. In some systems, the JAXB library is not added to the project sources by default. Right-click the **Libraries** folder under the **JAXBintro** project, select **Add Library…** and add the **JAXB 2.2** library from the list.
4. Add an element to the Schema that contains the book's publication year.

   You can edit the schema in a variety of ways. Simple modifications are best done manually form the "Source" view, but you can also use the visual editors in the "Schema" or "Design" views[1]. Browse the link **xmltools-exploreschema** on the exercise unit in the module site.
5. Now verify the modified Schema. Right-click the "Books.xsd" file and select the "Validate XML" option to validate the XML. The Output - XML check window should show that there are no validation errors.
6. Please note that once you generate the JAXB binding, a local copy of the schema file (.xsd) will be generated specifically for the binding. If you want any further changes you make to the schema file to be reflected in the generated Java classes, you need to modify the local copy of the schema file (in the project's **JAXB Bindings** folder), then right click **JAXB Bindings** folder to ***regenerate the java code***.

Part II Using Java classes to access XML Documents

1. Right-click the "jaxbIntro" project node and choose New → Other → XML → JAXB Binding. Then click Next.

   In the JAXB Binding wizard, edit the following:

   a) Type "BookRequest" in **Binding Name** and "itemOrder" in **Package Name**.

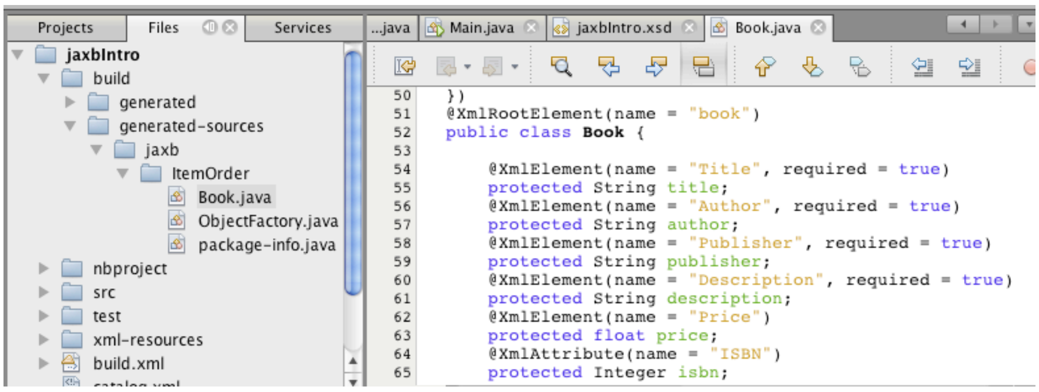
   b) Next to **Select From Local File System**, click **Browse** and browse to the XML Schema file that you modified above "Books.XSD".

   c) In the **Schema Type** should be "XML Schema" by default. Leave all the other default settings and Click Finish.

   The IDE generates the Java objects from the given XML document. The generated files are in the "build" folder.

[1] The XML visual editor is available as a plug-in. If you install netbeans on your own computer, you'll need to manually install it: http://blogs.oracle.com/geertjan/entry/xml_schema_editor_in_netbeans

2. Open the “Files” panel and then you can browse to the location of the **generated** Java objects (see the image below). Examine the **BookType** class in the “Book.java” file and note the methods



```
})
@XmlRootElement(name = "book")
public class Book {

    @XmlElement(name = "Title", required = true)
    protected String title;
    @XmlElement(name = "Author", required = true)
    protected String author;
    @XmlElement(name = "Publisher", required = true)
    protected String publisher;
    @XmlElement(name = "Description", required = true)
    protected String description;
    @XmlElement(name = "Price")
    protected float price;
    @XmlAttribute(name = "ISBN")
    protected Integer isbn;
```

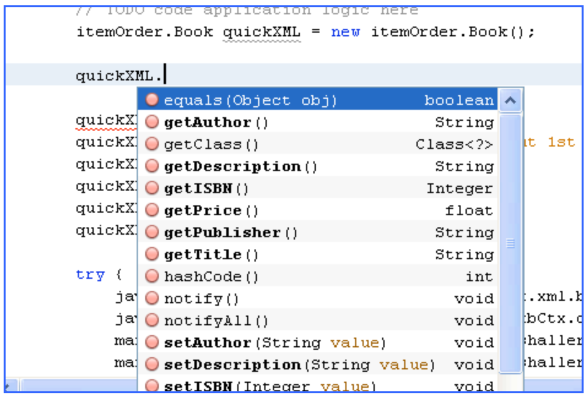
generated to get/set the XML elements.

3. Now you should uses the generated classes to create (marshal) XML output that is compatible with the base Schema.
   i. Go back to the “Project” window and open the “Main” class under file under (source Packages → jaxbintro).
   ii. Declare “itemOrder.Book”, which is one of the generated root JAXB classes, in the constructor:

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

            itemOrder.Book quickXML = new itemOrder.Book();
```

   iii. Now just type “quickXML.”, to start using your declaration, and then notice that when you press Ctrl-Space, the IDE gives you relevant code completion for your JAXB artefacts:



```
itemOrder.Book quickXML = new itemOrder.Book();

quickXML.

equals(Object obj)                  boolean
getAuthor()                         String
getClass()                          Class<?>
getDescription()                    String
getISBN()                           Integer
getPrice()                          float
getPublisher()                      String
getTitle()                          String
hashCode()                          int
notify()                            void
notifyAll()                         void
setAuthor(String value)             void
setDescription(String value)        void
setISBN(Integer value)              void
```

   iv. Set some values for the JAXB class, such as the following:

```
    quickXML.setAuthor("Lev Tolstoi");
     quickXML.setDescription("Russion fixction about 1st world war");
     quickXML.setISBN(62129985);
     quickXML.setPrice((float)12.6);
     quickXML.setPublisher("Progress");
     quickXML.setTitle("War and Peace");
```

v. Type the letters '**jaxbm**' into the editor. These letters stand for 'JAXB Marshalling'. You should now see the following:

```
jaxbm
```

A red underline appears, because the characters you typed do not form a word that is part of the Java programming language. Instead, these letters form a NetBeans code template, which we will use in the next step.

Press the '**Tab**' key.

The 'jaxbm' characters expand and a code snippet appears:

```
try {
    javax.xml.bind.JAXBContext jaxbCtx = javax.xml.bind.JAXBContext.newInstance(quickXM.getClass().getPackage().getName());
     javax.xml.bind.Marshaller marshaller = jaxbCtx.createMarshaller();
     marshaller.setProperty(javax.xml.bind.Marshaller.JAXB_ENCODING, "UTF-8"); //NOI18N
     marshaller.setProperty(javax.xml.bind.Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
     marshaller.marshal(quickXML, System.out);
```

vi. Run the application. The Output window should show a well-formed XML output based on the input schema.

vii. Modify the marshalling code in "Main.java" to write the output to an XML file instead of "System.out".

viii. Create another main class "readOrder.java" that similarly uses JAXB to read (unmarshalls) the content of the XML file generated at the previous step. Here you can similarly use code template for 'JAXB Unmarshalling': use '**jaxbu**' instead of 'jaxb' above and press the 'tab' key.

Part III Working with Lists of items in XML

1. In XML, you use 'unbounded occurrences' in a sequence to represent dynamic list of items. For instance, you can represent a collection of movies in a schema as follows:

   `<xsd:element name="Movies" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>`
   This is translated into a java Array List by JAXBM into: `List<String> movieCollection;`

2. In the module's content area, you will find a sample source code in (**JAXBadv.zip**) that demonstrates the use of Java Lists to process XML objects. The schema file sets up a complex type '`movieType`' and creates a sequence of an unbounded number of this type in '`movie_collection`'.
3. Create XML binding for the JAXBadv project similar to II.1 above. Use "MovieRequest" for the **Binding Name**, browse to "Shows.xsd" to select the **Schema File**, and type "Media" for the **Package Name**. *(Use exact spelling/case as the pre-written code expects these details)*
4. Clean and Build the project.
5. Examine the generated java code (see how in **II.**2 above) to understand the relationship between the root element '`ShowingToday`', the List '`MovieCollection`', and the object type '`MovieType`'.
6. The code in the '`main.java`' file sets up a root element, populates it with three elements (movies) and marshals it to a file. It also demonstrates how to iterate through the collection (java list) to process its elements.
7. To Do: Write another new java class that unmarshalls the XML document from the (Now_Showing.txt) file and selects for output films produced by 'Robert Benton'.

   *Hint: copy the imports from 'main.java', then add a declaration of the XML root element 'ShowingToday' and then use '**jaxbu**' as in 3.viii above to generate the unmarshalling code.*