AIXM Viewer
Implementation

By: Mark Austin, Natasha Shmunis, Michael Ball, University of Maryland, College Park, MD 20742. E-mail: [austin,kosnat]@isr.umd.edu; mball@rhsmith.umd.edu
Table of Contents

- Determining AIXM Viewer requirements
- Choosing the right process and tools
- Creating a simple prototype
- Identifying and resolving potential problems
In practice, AIXM Viewer should be able to read xml files, display the content including the AIXM features and properties. It also must be able to manipulate that Information.

As a first cut, the reader must read AIXM files, which are based on gml 3.x (xml format). These files have to conform to an AIXM schema.
Solutions

- Write your own parser to read in and write xml files and your own wrapper (the viewer).

- Not practical and time consuming. AIXM specification consists of thousand of objects.

- Find an automated way to read xml files based on provided XML Schema.

- More challenging.
Finding the tools: JAXB

- JAXB Java Architecture for XML binding.
  - Binds XML Schema to a relevant source code. It provided several good features:
  - The Unmarshaller provides the client application the ability to convert XML data into a tree of Java content objects. As a result the programmer does not have to write all the java code for visualization of xml objects.
  - The Marshaller provides the client application the ability to convert a Java content tree back into XML data. Therefore, it can automatically save java objects into the xml format.
AIXM Viewer with JAXB

- The steps for creating an AIXM Viewer would include:
- Converting AIXM Schema into java source classes and objects with JAXB.
- Creating wrapping classes for visualization and manipulation of the AIXM messages.
- Incorporating GML viewer’s functionality.
<?xml version="1.0" encoding="UTF-8"?>
<shapes
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='file:/C:/FAA/shapeviewer/shapes/Shapes.xsd'>
  <shape name = "circle" id ="1">
    <x1>10</x1>
    <y1>10</y1>
    <x2>100</x2>
    <y2>100</y2>
  </shape>
  <shape name = "square" id ="3">
    <x1>120</x1>
    <y1>10</y1>
    <x2>100</x2>
    <y2>100</y2>
  </shape>
  <shape name = "line" id="2">
    <x1>100</x1>
    <y1>100</y1>
    <x2>200</x2>
    <y2>200</y2>
  </shape>
</shapes>
Unmarshalling

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema

xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="shapes" type = "shapesType"/>
    <xsd:complexType name = "shapesType">
        <xsd:sequence>
            <xsd:element name ="shape"
                minOccurs="unbounded">          
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name ="x1" type ="xsd:int"/>
                        <xsd:element name ="y1" type ="xsd:int"/>
                        <xsd:element name ="x2" type ="xsd:int"/>
                        <xsd:element name ="y2" type ="xsd:int"/>
                    </xsd:sequence>
                    <xsd:attribute name = "name" type="xsd:string"/>
                    <xsd:attribute name = "id" type="xsd:int"/>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

```java
public static class Shape {
    protected int x1;
    protected int y1;
    protected int x2;
    protected int y2;
    @XmlAttribute
    protected Integer id;
    @XmlAttribute
    protected String name;

    public int getX1() {
        return x1;
    }
    public void setX1(int value) {
        this.x1 = value;
    }

    public int getY1() {
        return y1;
    }
    public void setY1(int value) {
        this.y1 = value;
    }
    ...
    ...
    ...
}
```
Unmarshalling

- Opening an XML file requires three easy steps.

```java
// After the source files are generated from the schema,
// unmarshall them into a memory tree
JAXBContext jc = JAXBContext.newInstance("shapeviewer.shapes.generated");
Unmarshaller u = jc.createUnmarshaller();

// Get objects from the tree
JAXBElement element = (JAXBElement) u.unmarshal(file);
ShapesType shapes = (ShapesType)element.getValue();

// Pass objects into the panel for drawing
shapeList = shapes.getShape();
panel.setShapeList(shapeList);
```
Challenges

- AIXM Schema needs to be changed in order to be compatible with JAXB technology.
- Huge XML files are usually a problem.
- How to convert a small prototype into a full feature application.
Conversion of the schema specifications into Java classes imposes programming challenges.

For example: JAXB does not know how to handle objects derived by extension from objects derived by restriction.

This is a logical problem that can be fixed only by restructuring the schema.
AIXM Schema

```xml
<complexType name="AbstractAIXMFeatureBaseType" abstract="true">
    <complexContent>
        <restriction base="gml:DynamicFeatureType">
            <sequence>
                <element ref="gml:description" minOccurs="0"></element>
                <element ref="gml:name" minOccurs="0" maxOccurs="unbounded"></element>
                <element ref="gml:boundedBy" minOccurs="0"></element>
            </sequence>
            <attribute ref="gml:id" use="required"></attribute>
        </restriction>
    </complexContent>
</complexType>

<complexType name="AbstractAIXMFeatureType" abstract="true">
    <complexContent>
        <extension base="aixm:AbstractAIXMFeatureBaseType">
            <sequence>
                <group ref="aixm:StandardAIXMFeatureProperties"></group>
                <group ref="aixm:DynamicFeatureProperties"></group>
                <element name="featureMetadata" type="aixm:FeatureMetadataPropertyType" minOccurs="0"></element>
            </sequence>
        </extension>
    </complexContent>
</complexType>
```