Worcester Polytechnic Institute

ASSISTments Interactive Qualifying Project

Prepared by
Thinh Nguyen
M.S. Douglas Selent

Prepared for
Professor Neil Heffernan

Date
29 April 2015
Abstract

This Interactive Qualifying Project (IQP) focuses on XML binding and Data Dumping. First, XML binding project assists transferring data among databases as well as providing ASSISTments data structure in form of XML to them. Second, Data Dumper concentrates on serving researches’ request on particular ASSISTments’ data. We implemented Problem Level Report, Action Level Report and many other features to serve researchers’ demands.
Acknowledgement

I would like to special thank Professor Neil Heffernan for allowing me to work on the ASSISTments project. I appreciate the empathy he has shown towards me in respect to time consideration, by keeping our meetings timely and efficient; thus allowing me more time to complete this work. I would like to thank him for his efforts on helping me shape the content in this paper. This work would not be possible without his dedication and hard work. Also, I would like to send a thank you to Douglas Selent because he has been a great mentor throughout this IQP. I appreciate the time and effort that he put into guiding me throughout both projects. Lastly, I would like to thank Yang Lu for assisting me on the structure of Data Dumper and teaching me about ASSISTment’s data structure.
## Table of contents

1. Introduction
2. XML Binding Project
   2.1 What is XML?
   2.2 XML Elements
      2.2.1 Tag
      2.2.2 Element
      2.2.3 Attribute
      2.2.4 XML declaration
   2.3 Why choose XML?
   2.4 Writing XML Schema
   2.5 JAXB Overview
      2.5.1 What is JAXB?
      2.5.2 Why JAXB?
      2.5.3 Java Environment Setup
      2.5.4 JAXB syntax
         2.5.4.1 XML Schema
         2.5.4.2 Elements of XML Schema
   2.6 JAXB Marshalling and Unmarshalling
      2.6.1 Generating Java class from JAXB
   2.7 JAXB Annotation
   2.8 How to generate JAXB bindings

3. Data Dumper
   3.1 Removing duplicate columns in SQL query
      3.1.1 DataSet
      3.1.2 DataSet Row Mapping
      3.1.3 Dumping Problem Level Report
   3.2 T-Test Function
   3.3 Adding columns to DataSet
   3.4 Comparing Problem DataSet
   3.5 Action Level Report
   3.6 Unit tests

4. Conclusion
5. References

Appendix A
1. Introduction

The first goal of this Interactive Qualifying Project (IQP) was to create a complete set of XML documents and schemas to represent ASSISTment’s data structure. The XML schemas were used for XML binding and Java classes generation. As a result, generated XML files and Java classes were used for two purposes. The first purpose was to transfer data from one database to another. The second purpose was to provide information of the problem set structure for researchers. Having the XML data, researchers can modify it and send back to the system to achieve further results for researching purposes.

Additionally, we added functions to the Data Dumper, an application that sends ASSISTment’s data to researchers for research purposes. In this IQP, we added several features to the Data Dumper such as T-Test, Action Level Report, Problem Level Report, Problem Copy Modification detection and Column Mapping. By having these functions, researchers are equipped with utilities when dumping ASSISTment’s data.

2. XML Binding Project

2.1 What is XML?

XML stands for Extensible Markup Language originally proposed by Bray et al in 1998 (Bray et al, 1998). Their goal was to create an easy-to-use language that is both machine and human readable over the internet. It is used to display data in a structured text-based format. An example of a car data represented in XML is

```xml
<?xml version="1.0"?>
<Car>
  <Make>Nissan</Make>
  <Model>Rogue</Model>
  <Year>2015</Year>
</Car>
```

Figure 1. Example of an XML document

2.2 XML Elements

An XML document contains the following elements

2.2.1 Tag

A markup construct begins with the left angle bracket “<” and ends with the right angle bracket “>”. There are three types of tags
Start tag. For instance `<tag>`
End tag. For instance `</tag>`
Empty element tags. For instance `<tag />`

Note that tags can be nested to create a hierarchy structure.
2.2.2 Element
An XML document part begins with a start tag and ends with a matching end tag. For instance,

```xml
<Print>Hello world</Print>
```

Figure 2. Example of a valid XML element

is a legitimate XML document. In this example, Hello world is considered as an element surrounded by <Print> </Print> tags.

2.2.3 Attribute
Attributes are used to specify the metadata of a tag. For instance, the example above, `<Print>Hello world</Print>`, can contain the color of Hello world element by having a color attribute. As a result, we have

```xml
<Print color="yellow">Hello world</Print>
```

Figure 3. Example of an XML document with attribute color

This example indicates that Hello world text has color yellow.

2.2.4 XML declaration
At the start of an xml document, there may be some information describing the document. For instance,

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

This tag indicates the xml’s version and type of text encoding.

2.3 Why choose XML?
We decided to choose XML because of its flexibility to serve a diverse need. Regardless of applications’ interfaces, programmers can share XML formats and apply it to their own applications’ structures. As a result, XML offer its users simplicity, extensibility, interoperability, openness and professional experience (Laurent 1998). We also chose to use XML because it is well-known and in a human-readable format. In addition to these properties, Java and the eclipse IDE offers support for working with XML files with the JAXB libraries.

**Simplicity**
It brings an intuitive environment to both programmers and researchers (Laurent 1998). Developers can pick up XML rules quickly as they are quite simple. Despite its simplicity, xml structure is solid with both human and machine readable. It can be written in nested structure to represent all kind of complex data while still maintain scalability and readability.

**Extensibility**
XML allows developers or researchers to insert new tags in the document without breaking the previous structure of the xml (Laurent 1998).

**Interoperability**
Because of its consistent structure, XML can be used across different tools and languages. XML parsers and interpreters usually are considered relatively low cost to build in many programming languages (Laurent 1998). Additionally, as XML support different character encoding standards, it can be used to encode complicated characters or languages across computer platforms.

**Openness**

Even though there are opinions on a standard procedure for creating XML, the XML’s structure itself is freely accessible and used through the world wide web (Laurent 1998).

**Professional Experience**

W3C specifies the standards for XML in a professional manner. The XML structure is widely understand and accepted by professional developer as one of the standards for representing data.

2.4. **Writing XML Schema**

For this project I created the XML schema’s and corresponding JAXB class files to represent Problem Sets, Problems, and Hints inside the ASSISTments system. Please consult Appendix A for all the XML schemas.

2.5. **JAXB Overview**

2.5.1 What is JAXB?

Java Architecture for XML Binding (JAXB) lets Java developers to represent Java classes in XML representations (Ort 2003). It also eases the process of accessing XML document in a Java development environment. JAXB contains two main features: the ability to map (marshal) Java object into XML and unmarshal XML format into Java object.

2.5.2 Why JAXB?

There are many XML marshalling libraries for Java. Woodstox (n.d.), xstream (n.d.), and Simple (n.d.) XML parser are few names that we can mention. However, we decided to go with JAXB because of its reliability and compatibility.

JAXB is reliable as it has been tested and developed by the Java Community Process, a formalized mechanism that allows interested parties to develop standard technical specifications for Java technology (Project JAXB 2013). JAXB 1.0 was released under the Java Specification Requests (JSRs) 31, and JAXB 2.0 was released under JSR 222 and Maintenance Release 2 in December 2009.

Additionally, the library is very compatible with the tools we are using at the lab. Firstly, it is built in Eclipse Integrated Development Environment (IDE) for class auto generation. Second, as we are using Java and XML, JAXB would be a perfect fit for the job. It has features that we are looking for: marshaling, unmarshaling, and XML schema binding. It would help us to levitate the process of translating java object to XML and vice versa.
2.5.3 Java Environment Setup

Setting up the java environment for JAXB is quite easy. As JAXB is built into Eclipse EE IDE, the easiest way to quickly interact with this library is to download Eclipse EE at http://eclipse.org/downloads/. Eclipse IDE would take care of all your java setting and environment. If you want try out from scratch, open Eclipse IDE. Click File -> New -> Java Project. Create a java file in src folder and begin to explore JAXB from there.

2.5.4 JAXB syntax

2.5.4.1 XML Schema

Before getting into Java programming for JAXB, we would like to introduce a concept that will be essential for XML marshalling. It is XML schema.

According to W3Schools, XML schema is an XML document that specifies the structure and constraints on the content of another XML document (XML Schema Tutorial, n.d.). These constraints often include the data type, tag name, and nested structure specifications. For instance, below is an XML schema for the car XML data in figure 1

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.w3schools.com"
    xmlns="http://www.w3schools.com"
    elementFormDefault="qualified">
    <xs:element name="Car">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Make" type="xs:string"/>
                <xs:element name="Model" type="xs:string"/>
                <xs:element name="Year" type="xs:integer"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 4. Car XML Schema

2.5.4.2 Elements of XML Schema

XML schema has a large number of tags and attributes to remember. However, in this project, there are only several key components that we used to specify an XML schema

The `<schema>` tag

This is the root element of every XML schema (XST, n.d.). For instance

```xml
<?xml version="1.0"?>
<xs:schema>
</xs:schema>
```

The tag may contain some attributes. Some are
xmlns:xs = http://www.w3.org/2001/XMLSchema
indicates that the elements and datatypes used in the schema come from
http://www.w3.org/2001/XMLSchema namespace.
targetNamespace=“url”
specifies that the elements defined by this schema come from the “url”
Additionally,
xmlns=http://www.w3schools.com
tells us that the default namespace is http://www.w3schools.com
Finally, this fragment
elementFormDefault=“qualified”
signifies that elements used by the XML document instance must be namespace qualified. In short,
qualified means that elements and attributes are in the targetNamespace of the schema
(Microsoft, n.d.)
unqualified indicates that elements and attributes do not have a namespace (Microsoft, n.d.)
Notes that elementFormDefault does not affect types’ namespace in the schema. It only
influences elements’ namespaces.
For instance,
<schema
 xmlns=“http://www.w3.org/2001/XMLSchema”
 targetNamespace=“url”
 />
<element name=“firstname” type=“xs:string”
 form=“qualified”></element>
<element name=“lastname” type=“xs:string”></element>

This example will put firstname element in the target namespace while specify null namespace for lastname element. Having elementFormDefault attribute will put all element into the target namespace by default unless an element specifically indicate a form=“unqualified” property. For instance,
<schema
 xmlns=“http://www.w3.org/2001/XMLSchema”
 targetNamespace=“url”
 elementFormDefault=“qualified”
 />
<element name=“firstname” type=“xs:string”></element>
<element name=“lastname” type=“xs:string”></element>

Both firstname and lastname elements are in the target namespace now.
The simple element
A simple element defines name of an XML tag and its element’s data type (XML Schema Tutorial, n.d.). For instance

```xml
<xs:element name="model" type="xs:string"/>
```
defines a tag named model and its data type is string. An associated XML for this schema is
```
<model>Nissan</model>
```
You can see that the datatype of this model tag is string since `xs:string` was specified. XML schema has a lot of built in data types. However, the most common types are:

- `xs:string`
- `xs:decimal`
- `xs:integer`
- `xs:boolean`
- `xs:date`
- `xs:time`
- `xs:long`

Attribute
According to W3schools, a simple element cannot have attributes (XML Schema Tutorial, n.d.). Instead, when an element has attributes, it is considered a complex type. For instance
```
<xs:element name="employee">
  <xs:complexType>
    <xs:attribute name="employeoid" type="xs:integer"/>
  </xs:complexType>
</xs:element>
```

Default and Fixed Values for Attributes
Attribute can have default or fixed value (XST, n.d.).
- **Default value** is assigned to an attribute when no other value is specified
- **Fixed value** is presumably assigned to an attribute and unchangeable.
For example
```
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
```

Optional and Required Attributes
Attribute may optionally appear in an element. To define this behavior, we can use use="required" or use="optional". By default, attributes are optional. For instance
```
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Restriction on value
Restriction tags restrain values for XML elements or attributes. They are also called facets.
For instance, to restrict a range from 0 to 60, we can specify it as

```xml
<xs:restriction base="xs:integer">
   <xs:minInclusive value = “0”/>
   <xs:maxInclusive value="120”/>
</xs:restriction>
```

For more examples about restriction, you can visit W3schools examples [here](#).

**Complex Element**

A complex element is an XML element that contains other elements and attributes. According to W3Schools, there are four types of complex elements:
- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Examples of Complex Element

**Empty elements**

```xml
<process id="3232"/>
```

For instance, the corresponding XML schema is

```xml
<xs:element name="process">
   <xs:complexType>
      <xs:attribute name="id" type="xs:integer"/>
   </xs:complexType>
</xs:element>
```

**Containing other elements**

```xml
<student>
   <firstname>John</firstname>
   <lastname>Doe</lastname>
   <age>14</age>
</student>
```

The XML associates with this XML schema

```xml
<xs:element name="student">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="firstname" type="xs:string"/>
         <xs:element name="lastname" type="xs:string"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```
<xs:element name="age" type="xs:integer"/>
</xs:sequence>
</xs:complexType>
</xs:element>

**Containing only text**

<vehicle type="car"> Nissan Rogue</vehicle>

This type contains only simple content (text and attributes); therefore, we add a simpleContent element around the content simpleContent tags around the content. The simpleContent tags have to go with an extension or a restriction within it. For instance

<xs:simpleContent>
<xs:extension base="type"/>
</xs:extension>
</xs:simpleContent>

OR

<xs:simpleContent>
<xs:restriction base="type"/>
</xs:restriction>
</xs:simpleContent>

**Containing both elements and text**

<action>
Time: <date lang="Norwegian">04.02.2015</date>
</action>

To implements a mixed content (attributes, elements, and text), we must specified the mixed attribute in the XML schema. For example

An XML document:
<action>
    Action time: <time> 313213 </time>
</action>

An XML schema associates with this XML document is
<xs:element name="action">
    <xs:complexType mixed="true">
<xs:sequence>
  <xs:element name="time" type="xs:long"/>
</xs:sequence>
</xs:complexType>
</xs:element>

**Indicators**

They are used to control elements in documents.
Overall, there are seven of them. They are distributed into three categories: order indicators, occurrence indicators, and group indicators

**Order indicators**

The <all> indicates that child elements can occur in any order with only once appearance. For instance

```xml
<xs:element name="person">
  <xs:complexType>
    <xs:all>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

**Choice indicator**

The <choice> tag indicates an alternation among elements. Only one value will be accepted.

```xml
<xs:element name="animal">
  <xs:complexType>
    <xs:choice>
      <xs:element name="fourlegs" type="type"/>
      <xs:element name="twolegs" type="type"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

**Sequence indicator**

The <sequence> tag determines that child elements must follow a specified order

```xml
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
```
<xs:element name="firstname" type="xs:string" />
<xs:element name="lastname" type="xs:string" />
</xs:sequence>
</xs:complexType>
</xs:element>

**Occurrence Indicators**
We often used `<minOccurs>` and `<maxOccurs>` tags in this project. `<minOccurs>` specifies the minimum number of times an element can occur. Similarly, `<maxOccurs>` indicates the maximum number of times an element can occur. These tags can be used to enforce multiplicity constraints on the elements in the XML file.

For instance
<xs:element name="Employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string" />
      <xs:element name="previouscompany" type="xs:string"
        maxOccurs="10" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

In this example, the minOccurs="0" indicates that the element can appear greater than zero time. Zero time means there is no element of this type. Furthermore, maxOccurs="10" specifies that this element can appear 10 times maximum. In short, the occurrence of previouscompany element is \(0 \leq x \leq 10\), with \(x\) is the number of times it appears.

**2.6 JAXB Marshalling and Unmarshalling**

The previous section introduces the XML schema, which is a way to restrain XML documents. Because we need to use JAXB in our project for translating XML documents to Java Object and vice versa, employing the XML schema was essential to specify how those XML documents should look like.

In this section, we will talk briefly of how to use JAXB to generate Java classes from XML schema. Additionally, we will introduce how to use JAXB marshalling and unmarshalling function.
2.6.1 Generating Java class from JAXB

Suppose that we have a schema Car.xsd

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Car">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Id" type="xs:integer"/>
        <xs:element name="Brand" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

which represents an XML document

```xml
<Car>
  <Id>232</Id>
  <Brand>Nissan</Brand>
</Car>
```

We would like to have a java class that corresponds to this XML schema. Traditionally, we manually create a java class with all the required fields. However, the Eclipse IDE has a tool to do it automatically for us. In other to create a Java class from XML schema, in Eclipse, we right clicked on Car.xsd > Generate > JAXB Classes. As a result, Eclipse will create a package named generated and put Car.java class in that package. Below is Car.java class.

```java
package;

import java.math.BigInteger;
import javax.xml.bind.annotation.*;

@AccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "id",
    "brand"
})
@XmlRootElement(name = "Car")
public class Car {
```
Here is a graphical step by step instruction of how to generate Java classes from XML schema. The example is on problem_set.xsd file.

**Step 1.**
Right click on the schema file and choose generate JAXB classes.
Step 2.
Choose a destination project folder.
Step 3.
Choose a source folder, destination package for the generated classes, and the binding file(s) to use.
Step 4.
Click finish to generate the classes or click next to specify additional options. The console output will show the generated files or any errors that occurred.

2.7 JAXB Annotation

When generating this Car.java class, JAXB put some annotation on the class and its field. These annotations help JAXB to understand the mapping between Java class and XML document. For instance, consider a Car.java class with JAXB annotation.

```java
@XmlElement(name="Car")
@AccessorType(XmlAccessType.Field)
@XmlElement(propOrder = { })
public class Car {
    @XmlElement(name="vin")
    private String vin;
    @XmlElement(name="brand")
```
private String brand;
@XmlElement(name="model")
private String model;

Figure 5. Example of JAXB annotations for XML binding

@XmlElement, @XmlAccessorType, @XmlType, and @XmlElement are considered JAXB annotations for XML binding. These annotations help JAXB to correctly bind fields or methods of a Java class to corresponding XML document’s elements.

@XmlAccessorType(XmlAccessType.FIELD)
This annotation defines the XML document’s access mechanism (Sun, n.d.). Usually, the value is specified to XmlAccessType.Field that indicates the mapping of this class’s fields and a corresponding XML document. Another value type is XmlAccessType.Property that allows JAXB to access class’s methods instead of class’s fields. For instance,

Example 1: XmlAccessType.Field
@XmlElement(name="Car")
@XmlAccessorType(XmlAccessType.Field)
public class Car {
    @XmlElement(name="brand")
    private String brand;

    public String getBrand() {
        return this.brand;
    }
}

Example 2: XmlAccessType.Property
@XmlElement(name="Car")
@XmlAccessorType(XmlAccessType.Property)
public class Car {
    private String brand;

    @XmlElement(name="brand")
    public String getBrand() {
        return this.brand;
    }
}
Both examples had the same field and method. However, the placement of @XmlElement is different in both examples. In example 1, @XmlElement is placed on the field; in example 2, @XmlElement is located at the class’ method.

```java
@XmlElementType(name = "", propOrder ={
    "id", "brand"
})
```

This annotation specifies which xml schema type this class can be mapped to (Sun, n.d.). name="" indicates that the class can be mapped to an anonymous schema type; otherwise, the class name maps to a complex type name.

Also, we can see there is a propOrder property in @XmlElement annotation. According to JAXB specification, this property can be used to specify the order of XML elements in xs:sequence. Since JAXB cannot keep track of the fields order in a java class, propOrder is used to manually indicate this order.

```java
@XmlElementRootElement(name = "Car")
```

Again, this annotation maps a class or an enum type to an XML element (Sun n.d.). It can be used with two program elements
- a top level class
- an enum type

According to JAXB documentation, when this annotation is specified for a top-level class or an enum type, its value is presented as XML element in an XML document. For example

```java
@XmlElementRootElement(name="PriceElement")
public class USPrice {
    @XmlElement
    public BigInteger price;
}
```

corresponds with XML schema

```xml
<xs:element name="PriceElement" type="USPrice"/>
    <xs:complexType name="USPrice">
        <xs:sequence>
            <xs:element name="price" type="xs:integer"/>
        </sequence>
    </complexType>
</xs:complexType>
```

and XML document

```xml
<USPrice>
```
<price> 10 </price>
</USPrice>

@XmlElement(name = "Id", required = true)
   According to JAXB specification, this annotation maps a JavaBean property to a XML element, which derived from property name (Sun, n.d.).

This annotation can be applied to the following elements:
- a JavaBean property
- non static, non transient field
- within XmlElements

However, in this project, we mostly use this annotation for mapping java class’ fields. Below is a simple example of this annotation’s usage

public class Student {
   @XmlElement(name=“name”)
   public String name;
}

corresponds with the following XML schema

<xsd:complexType name=“Student” />
   <xsd:sequence>
      <xsd:element name=“name” type=“xsd:string” />
   </xsd:sequence>
</xsd:complexType>

ObjectFactory.java
   When using JAXB tool to generate Java classes, the tool will also generate an ObjectFactory class that responsible for creating those generated classes. For example, the car.java class above generated an ObjectFactory class

package generated;

import javax.xml.bind.annotation.XmlRegistry;

@XmlRegistry
public class ObjectFactory {

   public ObjectFactory() {


As all the class in generated package was automatically made, this Object Factory class would create an interface that the user can interact to without touching the generated classes.

### 2.8. How to generate JAXB bindings

When using XML there are times when the name of the attribute conflicts with a previously define name. Consider the following XSD schema code fragment shown below.

```xml
<xs:element name="terminator" minOccurs="1" maxOccurs="1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="params" minOccurs="0" maxOccurs="1">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
              <xs:complexType>
                <xs:simpleContent>
                  <xs:extension base="xs:string">
                    <xs:attribute type="xs:string" name="name" use="optional"/>
                    <xs:attribute type="xs:string" name="value" use="optional"/>
                  </xs:extension>
                </xs:simpleContent>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
<xs:attribute type="xs:string" name="type"/>
</xs:complexType>
</xs:element>

Highlighted in red is the attribute with the name “value”. The problem with this is that all simple content (tag highlighted in blue) automatically inherits a “value” field. This will result in the following error shown below when trying to generate the XSD classes.

parsing a schema...
[ERROR] Property "Value" is already defined. Use &lt;jaxb:property&gt; to resolve this conflict.
line 56 of
jar:file:/C:/Stuff/Assitstment2/DataExporter2/api.2/api.2/lib/jaxb-xjc.jar!/com/sun/xml/xsom/impl/parser/datatypes.xsd

In order to fix this error, custom JAXB bindings must be used. The following is a binding template for used for generating the XSD schema.

<bindings xmlns="http://java.sun.com/xml/ns/jaxb"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema"
version="1.0">
  <bindings schemaLocation="problem_set.xsd" version="1.0">
    <bindings node="/xs:element[@name='terminator']">
      <bindings node="/xs:element[@name='params']">
        <bindings node="/xs:element[@name='param']">
          <bindings node="/xs:attribute[@name='value']">
            <property name="ValueAttribute"/>
          </bindings>
        </bindings>
      </bindings>
    </bindings>
  </bindings>
</bindings>

In this template the value field is renamed to ValueAttribute.

3. Data Dumper

ASSISTments is used extensively for research purposes. A researcher, who wants to run a study in the ASSISTments system, first fills out a google spreadsheet with their name, email, a link to their study design, and other information related to the specific details of the study. If
their study has been accepted to run in the system, teachers can use the content with their students inside the ASSISTments system. More information on how to run and analyze randomized control trials in ASSISTments and receive the data can be found at the following links http://tinyurl.com/nhs7dd9, http://tinyurl.com/13ogm2m, and http://tinyurl.com/jvuy2ut. All the data from the study is stored in the ASSISTments database, which is infamously difficult to work with and analyze data.

The “Data Dumper” is a tool that researchers use to get reports on their data from studies that have run using the ASSISTments system. Instead of forcing researchers to learn how to use the ASSISTment’s database to analyze data, a large amount of work has been put into making the analysis process easier. Researchers use the Data Dumper to obtain several types of custom reports specifically designed to run statistical analysis on. These reports are in the form of CSV files, containing over fifty different information attributes. The Data Dumper system was created by Yang Lu, a masters student in computer science, and Douglas Selent, a Ph.D. student in computer science. In addition to the system structure and reporting, are several functions to transform the original dataset into different types of datasets.

Since a single dataset is not always the best for doing certain types of analysis, several other types of reports were generated based on the original data query. The original report included all the data at the problem level, where there is one row per problem. A student-level report, created by Korinn Ostrow, a learning science Ph.D. student, and Douglas Selent, is generated by transforming the original problem level report to have one row per student. A student/feature hybrid report is also generated from the problem level data by transforming it to have one row per student per feature. The combination of all these reports, allow researchers to mine several aspects of the data related to their study.

My goals were to extend the Data Dumper project to further automate the analysis process of ASSISTments data. There were several tasks related to this project that I worked on. My first task was to remove duplicated columns in the SQL query in order to get familiar with the Data Dumper code. My second task was to make the internal data structure, for storing the data, more robust and capable of more functionality. My third and most important task was to provide functionality to run statistical tests on columns in the data set. The two prior tasks lead up to this larger goal of automatic statistical testing functionality. Making the internal data structure more robust also allowed me to detect duplicated problem sets, which can be included in the reports. Lastly I implemented a new report called the Action Level report to help researchers analyze data at the actions level.

3.1 Removing duplicate columns in SQL query

When querying Problem Level Report from the database, there were duplicated columns when we did inner join between tables. For example, when joining table 1 and 2 on column A, the result is a table that containing Table1.A and Table2.A. We saw this as a duplication since
we just need one instance of column A, so we tried to remove those duplicate columns in the result table. For instance.

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

Result table when doing a join between two tables on column A

<table>
<thead>
<tr>
<th>Result Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table1.A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>Table2.A</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

However, our desired table is the following

<table>
<thead>
<tr>
<th>Result Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

Therefore, in order to achieve what we want, we used USING keyword in SQL. For instance,

```
SELECT * FROM
Table1 JOIN Table2 ON Table1.A = Table2.A
```

becomes

```
SELECT * FROM
Table1 JOIN Table2 USING(A)
```

This simple change helped us to achieve what we wanted. We successfully removed all duplication in the SQL query.
3.1.1 Dataset
After removing duplicated columns in the result, we focused on implementing features for the data dumper. One of the first things that we implemented was Dataset, a class that represents how we store data, you can imagine Dataset as a table with headers row and multiple data rows as follow.

![Dataset UML Diagram]

Figure 6: Dataset UML
We aim to present most of our dumped data in Dataset form, so we can have a central point to export data out. Currently, a Dataset can depicted as a CSV file. This data can be send to researchers upon their requests.

3.1.2 Dataset Row Mapping
Because of the need to create dataset with our desired column names, there is a need to map the query result table’s columns to our custom dataset’s columns. In order to achieve this, we created DatabaseMapping class. This class is responsible for all the column mapping.
3.1.3 Dumping Problem Level Report

One of the most important piece of data in data dumper is Problem Level Report, which is a result from querying and joining multiple tables together. To export this result into dataset form, we implemented a method called getProblemLevelReport as follow.

```java
public DataSet getProblemLevelReport(int sequenceId) throws DataAccessException, SQLException
{
    String sql = readSQLFile("/sql/DumpProblemReport.sql");

    ProblemLevelCallableStatementCreator creator = new ProblemLevelCallableStatementCreator(sql, sequenceId);
    ProblemLevelCallableStatementCallback callback = new ProblemLevelCallableStatementCallback();

    DataSet problemLevelData = m_database.execute(creator, callback);

    return problemLevelData;
}
```

This method basically read in the sql query string and passed arguments to it (using ProblemLevelCallableStatementCreator). Also, it indicated columns mapping between query
result and a DataSet using ProblemLevelCallableStatementCallback. Finally, the dataset result is achieved through executing the above sql query string.

3.2 T-Test Function

Per request, there is a need to know how similar two columns in a DataSet are. To accomplish this, we implemented a T-Test method that takes in a dataset, two column indices for that dataset, and a boolean if the T-Test is paired. Several studies in ASSISTments have experiments with a control group and an experimental group. One common method to analyze the data is to use a T-Test on common measures (such as average correctness per student).

Paired T-Test

The paired t-test is generally used when measurements are taken from the same subject before and after some intervention (Caprette, n.d.). “For example, it can determine the significance of a difference in blood pressure before and after administration of an experimental pressor substance” (Caprette, n.d). You can also use a paired t-test to examine identical pair samples that are subjected to different conditions. “For instance, you might test the effectiveness of a water additive in reducing bacterial numbers by sampling water from different sources and comparing bacterial counts in the treated versus untreated water sample” (Caprette, n.d). As a result, each different water source would give a different pair of data points.

Below is the our T-Test method for DataSet class

```java
public static DataSet TTest(DataSet problemLevelData, int colIndex1, int colIndex2, boolean isPaired) throws Exception
{
    /* Some conversions */
    String[] column1 = problemLevelData.getColumn(colIndex1);
    String[] column2 = problemLevelData.getColumn(colIndex2);
    double[] sample1 = (double[]) ConvertUtils.convert(column1, Double.TYPE);
    double[] sample2 = (double[]) ConvertUtils.convert(column2, Double.TYPE);

    StringBuilder t_statistic = new StringBuilder("t-statistic: ");
    StringBuilder pValue = new StringBuilder("p-value: ");

    /* if two samples are not equal in size */
    if (sample1.length != sample2.length)
    {
        throw new Exception("Unequal sample sizes for t test");
    }
```
/* If this is a paired T-Test */
if (isPaired)
{
    /* calculate t-statistic and pValue */
t_statistic.append(TestUtils.pairedT(sample1, sample2));
pValue.append(TestUtils.pairedTTest(sample1, sample2));
}
else /* If it is NOT a paired TTest */
{
    SummaryStatistics summary1 = new SummaryStatistics();
    SummaryStatistics summary2 = new SummaryStatistics();

    /* because we checked that two sample are equal in size,
    we can combine them in this loop */
    for (int i = 0; i < sample1.length; i++)
    {
        summary1.addValue(sample1[i]);
        summary2.addValue(sample2[i]);
    }

    /* calculate t-statistic and pValue */
t_statistic.append(TestUtils.t(summary1, summary2));
pValue.append(TestUtils.tTest(sample1, sample2));
}

DataSet result = new DataSet();
result.addDataRow(new String[] { t_statistic.toString(),
pValue.toString() });
return result;

This T-Test method would return a dataset that contains the t-statistic and p-value, which researchers can report in their results.

3.3 Adding columns to DataSet

After implementing T-Test method, we realized that we needed a mechanism to represent column in our DataSet. Currently, a DataSet is a list of DataRow, which contains a list of string. However, instead of adding a list of DataColumn to represent columns in DataSet, we decided to reuse the list of DataRow as a pseudo way to add, remove and get columns. We implemented three methods addColumn(), removeColumn, and getColumn().

Below is the details of our implementation
public void addColumn(int columnIndex) {
    for (DataRow row : dataRows) {
        row.addNewColumnCell(columnIndex);
    }
}

public void removeColumn(int columnIndex) {
    for (DataRow row : dataRows) {
        row.removeColumnCell(columnIndex);
    }
}

public String[] getColumn(int columnIndex) {
    Integer size = dataRows.size();
    String[] column = new String[size];

    for (Integer i = 0; i < size; i++) {
        DataRow row = dataRows.get(i);
        column[i] = row.getColumnCell(columnIndex); //O(1)
    }

    return column;
}

As you can see, adding or removing column is simply iterating through a list of DataRow and modified it at a particular index. Similarly, getting a column adds the value at a particular index into an array of string. The worst-case running time for these operations is O(n), with n is the number of row.

3.4. Comparing Problem DataSet

It often occurs that problem sets in ASSISTments are copied but not changed. When analyzing studies, we would like to include the copied problem set in the analysis if and only if it is identical to the original. We need to compare the problem set structure of the original to the problem set structure of the copies to see if the original problem set was modified since the last time it was opened. In the Data Dumper, we aim to determine if the two problem sets were identical. Using our DataSet class structure, this problem becomes equivalent to determining if the two DataSet objects are equals. Therefore, we override equals methods in DataSet.java.

@override
public boolean equals(Object obj) {
    boolean result = true;

    if (obj == null) {
        result = false;
    } else if (this == obj) {
        result = true;
    } else if (!(obj instanceof DataSet)) {
        result = false;
    } else {
        DataSet other = (DataSet) obj;
        if (this.getSize() == other.getSize()) {
            /* Checking for the header row number of column*/
            if (this.getHeaderRow() != null && other.getHeaderDataRow() != null) {
                result = this.getHeaderRow().equals(other.getHeaderDataRow());
            }

            Integer size = this.getSize();
            for (Integer i = 0; i < size && result; i++) {
                result = this.getDataRow(i).equals(other.getDataRow(i))
                        && result;
            }
        } else {
            result = false;
        }
    }
    return result;
}

Figure 8. Override equals method for DataSet

We also need to override equals() method in DataRow since DataSet’s equals method requires comparing two DataRows

@override
public boolean equals(Object obj) {
    boolean result = true;

if (obj == null) {
    result = false;
} else if (this == obj) {
    result = true;
} else if (! (obj instanceof DataRow)) {
    result = false;
} else {
    DataRow other = (DataRow) obj;
    if (this.getColumn() == other.getColumn()) {
        int length = this.getColumn();
        for (int i = 0; i < length && result; i++) {
            result = result &&
                    this.getColumnCell(i).equals(other.getColumnCell(i));
        }
    } else {
        result = false;
    }
}

return result;

Figure 9. Override equals method for DataRow

After implementing equals methods for DataRow and DataSet, we wrote a class named ComparableProblemExtractor to extract appropriate columns of Problem Level DataSet. We wanted to compare those columns together to see if two Problem Level DataSets are identical.

public class ComparableProblemExtractor implements ResultSetExtractor<DataSet> {
    @Override
    public DataSet extractData(ResultSet rs) throws SQLException, DataAccessException {

        DataSet result = new DataSet();
        result.setHeaderRow(new DataRow(
            new String[] {
                "position", "depth", "type", "assistance_id"
            }
        ));

        return result;
    }
}
3.5. Action Level Report

The current reports that are already generated by the Data Dumper are the Problem Level Report, Student Level Report, and the Student/Feature Hybrid Report. ASSISTments stores all the information on a student’s actions for every given problem. These actions include timestamps, the student’s answer (correct or incorrect), and when the student resumed a problem, clicked a hint, or used scaffolding questions. In the ASSISTments database, all this information is stored in a single column as a gigantic string in the YAML format (Ben-Kiki 2009). Due to the poor data storage format, all researchers with the exception of (Duong, Zhu et al) have avoided using the action level data in analysis. I used the YAML parser to parse the action level data and created the Action Level Report, which researchers can easily use to analyze data at the action level (Snakeyaml n.d.).

To illustrate the difficulty in working with the student actions, Figure 10 shows an example of a single entry for the actions column in the database. This entry represents all the actions a student took on a single problem.

---
- - start
  - 23423423
- - hint
  - 2343
- - answer
  - 42343
  - True
Figure 10. Action level report for

The example in Figure 10 shows that a student started his/her problem at timestamp 23423423. This number is represented in POSIX time, or the time in milliseconds from January 1, 1970. He then opened hint 2.343 seconds after starting the problem and answered this problem at 23465766 (23423423+42343). The time of the action can be calculated by adding the time shown in the actions report to the most recent time of the start or resume action. Start and resume actions are represented with an absolute timestamp, where the other action timestamps represent the relative times or the time difference from the last last start or resume action. Continuing with this example, the student’s answer was x/24, and it was correct represented by the “True” text above the answer.

<table>
<thead>
<tr>
<th>Actions</th>
<th>Parameters</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>timestamp</td>
<td>33242343</td>
</tr>
<tr>
<td>hint</td>
<td>timestamp</td>
<td>24324324</td>
</tr>
<tr>
<td>answer</td>
<td>correctness</td>
<td>True/False</td>
</tr>
<tr>
<td></td>
<td>answer text</td>
<td>“Any text”</td>
</tr>
<tr>
<td>scaffold</td>
<td>timestamp</td>
<td>32342343</td>
</tr>
<tr>
<td>url</td>
<td>timestamp</td>
<td>34234324</td>
</tr>
<tr>
<td>survey_response</td>
<td>timestamp</td>
<td>32423423</td>
</tr>
<tr>
<td></td>
<td>survey response number</td>
<td>1 to 5</td>
</tr>
<tr>
<td>end</td>
<td>no parameters</td>
<td></td>
</tr>
</tbody>
</table>

After understanding action level report, we would like to parse this format into a spreadsheet for action report. The format of the spreadsheet is

<table>
<thead>
<tr>
<th>User ID</th>
<th>Problem ID</th>
<th>Problem Sub-Part ID</th>
<th>Order</th>
<th>Action Type</th>
<th>Timestamp</th>
<th>Answer Text</th>
<th>Correctness</th>
<th>Problem Logs ID</th>
</tr>
</thead>
</table>
Yaml format
The format of the action level report in database is Yaml. Therefore, in order to build the spreadsheet format we need, we parsed this Yaml format into a DataSet and exported it out as csv file. We employed Snakeyaml as our Yaml parser (Snakeyaml, n.d.). Some limitations of the parser include not being able to parse data stored in binary format and not being able to parse certain strings with quote characters in them.

3.6 Unit tests
A set of JUnit tests were built into datadumper to make sure implemented features function properly. Also, these JUnit tests will help to prevent future implementations to break old code in Data Dumper. For each Java class, we created a JUnit test cases.

4. Conclusion
For our XML binding project, we successfully created all the XML schemas for the system as well as corresponding classes and XML documents. This project has been playing an important role in transferring data from one database to another. As easily interact with JAXB for XML binding, we could keep the process of marshalling and unmarshalling Java objects, those containing database data, became fast and efficient.

Furthermore, we successfully implemented data dumper into the ASSISTment family. With datadumper, researchers can know request the data they want from ASSISTment to learn more about student’s behavior and performance. We implemented some utilities into Data Dumper to ease the researcher from trivial calculation or statistics. Researchers can now receive data they wanted when they wanted.

However, we have not fully implemented feature that allow researchers to resend XML documents back to ASSISTment for data interpretation and processing in XML binding project. The reason for this was that we were still in the brainstorming process to be able to deliver the correct implementation to researchers. In the future, we expect that this feature will come out and widely used by users.

Future Work
One of the accomplishments of this project was the creation of code to automatically perform various statistical tests on the data sets. However this was only half of what was needed to fully automate the analysis. In order to run the analysis, the condition groups that students were assigned into must be known. While the analysis code was being written, Yang Lu was also implementing a method to automatically determine the condition the student was assigned
into. Currently both the analysis code, and the condition detection have been completed. It is future work to connect the two pieces of code in order fully automate the experiment analysis.

Another possibility for future work is to allow researchers to tag the sections of the experiment. These taggings can then be used to do more specific analyses automatically.

5. References

   ult(v=vs.110).aspx

   http://yaml.org/spec/1.2/spec.html


   https://sites.google.com/site/assitmenttestbed/the-data/d-how-to-get-your-data

   Retrieved from


Appendix A

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="problem_set">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="revision" type="xs:long" minOccurs="0" maxOccurs="1"/>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="navigator" minOccurs="1" maxOccurs="1"/>
    </xs:complexType>
  </xs:sequence>
  <xs:element name="params" minOccurs="0" maxOccurs="1">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute type="xs:string" name="name" use="optional"/>
                <xs:attribute type="xs:string" name="value" use="optional"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
The list of child problems/problem sets can accept any of the following id's
1. UUID (long unique id)
2. external_reference (short user friendly id)
3. old_id (either in id form or string form) (just string form?)
Figure 11. Problem Set XML Schema

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="AnswersTable.xsd" />
  <xs:include schemaLocation="BuggyMessages.xsd" />

  <xs:element name="Answers" type ="Answers"/>
  <xs:complexType name="Answers">
    <xs:sequence>
      <xs:element ref="AnswersTable" minOccurs="0"/>
      <xs:element ref="BuggyMessages" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

Figure 12. Answer.xsd XML Schema

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
<xs:element name="AnswersTable" type ="AnswersTable"><xs:element>
<xs:complexType name="AnswersTable">
   <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="value" type="xs:string" minOccurs="0"/>
      <xs:element name="isCorrect" type="xs:boolean" minOccurs="0"/>
      <xs:element name="problemID" type="xs:long" minOccurs="0"/>
      <xs:element name="mediaID" type="xs:long" minOccurs="0"/>
      <xs:element name="position" type="xs:long" minOccurs="0"/>
      <xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
Figure 13. AnswersTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:include schemaLocation="AnswerTypesTable.xsd" />

   <xs:element name="AnswerTypes" type ="AnswerTypes"><xs:element>
   <xs:complexType name="AnswerTypes">
      <xs:sequence>
         <xs:element ref="AnswerTypesTable" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
Figure 14. AnswerTypes.xsd
Figure 18. BuggyMessages.xsd

```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  
  <xs:element name="BuggyMessagesTable" type ="BuggyMessagesTable"/>
  <xs:complexType name="BuggyMessagesTable">
    <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="answerID" type="xs:long" minOccurs="0"/>
      <xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 19. BuggyMessagesTable.xsd

```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  
  <xs:include schemaLocation="CompatabilityReferencesTable.xsd" />
  
  <xs:element name="CompatabilityReferences" type="CompatabilityReferences"/>
  <xs:complexType name="CompatabilityReferences">
    <xs:sequence>
      <xs:element ref="CompatabilityReferencesTable" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Figure 20. CompatabilityReferences.xsd
Figure 21. CompatabilityReferencesTable.xsd

Figure 22. ContentExternalReferences.xsd
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ContentExternalReferencesTable" type=
"ContentExternalReferencesTable"/>
  <xs:complexType name="ContentExternalReferencesTable">
    <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="partnerID" type="xs:long" minOccurs="0"/>
      <xs:element name="tinyContentID" type="xs:long" minOccurs="0"/>
      <xs:element name="revisionNumber" type="xs:long" minOccurs="0"/>
      <xs:element name="contentTypeID" type="xs:long" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
Figure 23. ContentExternalReferencesTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="ContentTypesTable.xsd" />
  <xs:element name="ContentTypes" type="ContentTypes"/>
  <xs:complexType name="ContentTypes">
    <xs:sequence>
      <xs:element ref="ContentTypesTable" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
Figure 24. ContentTypes.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ContentTypesTable" type="ContentTypesTable"/>
  <xs:complexType name="ContentTypesTable">
    <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="partnerID" type="xs:long" minOccurs="0"/>
      <xs:element name="tinyContentID" type="xs:long" minOccurs="0"/>
      <xs:element name="revisionNumber" type="xs:long" minOccurs="0"/>
      <xs:element name="contentTypeID" type="xs:long" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
<xs:element name="id" type="xs:long" minOccurs="0"/>
<xs:element name="tableName" type="xs:string" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
Figure 25. ContentTypesTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="HintsTable.xsd"/>
  <xs:element name="Hints" type="Hints"></xs:element>
  <xs:complexType name="Hints">
    <xs:sequence>
      <xs:element ref="HintsTable" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
Figure 26. Hints.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="HintsTable" type="HintsTable"></xs:element>
  <xs:complexType name="HintsTable">
    <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="value" type="xs:string" minOccurs="0"/>
      <xs:element name="medialID" type="xs:long" minOccurs="0"/>
      <xs:element name="position" type="xs:long" minOccurs="0"/>
      <xs:element name="tutorStrategyID" type="xs:long" minOccurs="0"/>
      <xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
Figure 27. HintsTable.xsd
Figure 28. ProblemDifficulties.xsd

Figure 29. ProblemDifficultiesTable.xsd

Figure 30. ProblemInfosTable.xsd
<xs:include schemaLocation="QualityLevels.xsd" />
<xs:include schemaLocation="AppearanceTypes.xsd" />

<xs:element name="ProblemInfos" type="ProblemInfos"></xs:element>
<xs:complexType name="ProblemInfos">
  <xs:sequence>
    <xs:element ref="ProblemInfosTable" minOccurs="0" />
    <xs:element ref="QualityLevels" minOccurs="0" />
    <xs:element ref="AppearanceTypes" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
</xs:schema>

Figure 30. ProblemInfos.xsd

<xs:element name="ProblemInfosTable" type="ProblemInfosTable"></xs:element>
<xs:complexType name="ProblemInfosTable">
  <xs:sequence>
    <xs:element name="id" type="xs:long" minOccurs="0" />
    <xs:element name="problemID" type="xs:long" minOccurs="0" />
    <xs:element name="qualityLevelID" type="xs:long" minOccurs="0" />
    <xs:element name="pageNumber" type="xs:long" minOccurs="0" />
    <xs:element name="problemNumber" type="xs:long" minOccurs="0" />
    <xs:element name="appearanceTypeID" type="xs:long" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
</xs:schema>

Figure 31. ProblemInfosTable.xsd

<xs:include schemaLocation="ProblemOwnershipsTable.xsd" />

<xs:element name="ProblemOwnerships" type="ProblemOwnerships"></xs:element>
<xs:complexType name="ProblemOwnerships">
  <xs:sequence>
<xs:element ref="ProblemOwnershipTable" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

Figure 32. ProblemOwnership.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="ProblemOwnershipTable" type="ProblemOwnershipTable"/>
<xs:complexType name="ProblemOwnershipTable">
<xs:sequence>
<xs:element name="id" type="xs:long" minOccurs="0"/>
<xs:element name="problemID" type="xs:long" minOccurs="0"/>
<xs:element name="owner" type="xs:long" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

Figure 33. ProblemOwnershipTable.xsd

<xs:include schemaLocation="ProblemsTable.xsd"/>
<xs:include schemaLocation="ProblemTypes.xsd"/>
<xs:include schemaLocation="ProblemOwnershipTable.xsd"/>
<xs:include schemaLocation="ProblemDifficulties.xsd"/>
<xs:include schemaLocation="AnswerTypes.xsd"/>
<xs:include schemaLocation="ContentExternalReferences.xsd"/>
<xs:include schemaLocation="ProblemToProblemSetAssociations.xsd"/>
<xs:include schemaLocation="ProblemInfos.xsd"/>
<xs:include schemaLocation="Answers.xsd"/>
<xs:include schemaLocation="TutorStrategies.xsd"/>

<xs:element name="Problems" type="Problems"/>
<xs:complexType name="Problems">
<xs:sequence>
Figure 34. Problems.xsd

Figure 35. ProblemSetInfos.xsd

Figure 36. ProblemToProblemSetAssociations.xsd
<xs:element name="problemSetID" type="xs:long" minOccurs="0"/>
<xs:element name="copiedFrom" type="xs:long" minOccurs="0"/>
<xs:element name="dateCopied" type="xs:dateTime" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
Figure 36. ProblemSetInfosTable.xsd

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="ProblemSetLinksTable.xsd" />
  <xs:element name="ProblemSetLinks" type="#ProblemSetLinks"/>
  <xs:complexType name="#ProblemSetLinks">
  
  </xs:complexType>
</xs:schema>
Figure 37. ProblemSetLinks.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="ProblemSetLinksTable" type="#ProblemSetLinksTable"/>
  <xs:complexType name="#ProblemSetLinksTable">

   
  </xs:complexType>
</xs:schema>
Figure 38. ProblemSetLinksTable.xsd
Figure 39. ProblemSets.xsd
<xs:complexType name="ProblemSetsTable">
  <xs:sequence>
    <xs:element name="id" type="xs:long" minOccurs="0"/>
    <xs:element name="contentExternalReferenceID" type="xs:long" minOccurs="0"/>
  </xs:sequence>
  <!--
  <xs:element name="oldID" type="xs:long" minOccurs="0"/>
  -->
  <xs:element name="name" type="xs:string" minOccurs="0"/>
  <xs:element name="description" type="xs:string" minOccurs="0"/>
  <xs:element name="typeID" type="xs:long" minOccurs="0"/>
  <xs:element name="parentProblemSetID" type="xs:long" minOccurs="0"/>
  <xs:element name="qualityLevelID" type="xs:long" minOccurs="0"/>
  <xs:element name="grade" type="xs:string" minOccurs="0"/>
  <xs:element name="parameters" type="xs:string" minOccurs="0"/>
  <xs:element name="owner" type="xs:long" minOccurs="0"/>
  <xs:element name="allProblemsChildren" type="xs:boolean" minOccurs="0"/>
  <xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>
  <!--
  <xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
  <xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
  -->
</xs:complexType>
</xs:schema>

Figure 40. ProblemSetsTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xsi="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="ProblemSetTypesTable.xsd"/>

  <xs:element name="ProblemSetTypes" type="ProblemSetTypes"/>
</xs:schema>
Figure 41. ProblemSetTypes.xsd

Figure 42. ProblemSetTypesTable.xsd

Figure 43. ProblemsTable.xsd
<xs:element name="randomizeAnswers" type="xs:boolean" minOccurs="0"/>
<xs:element name="metadata" type="xs:string" minOccurs="0"/>
<xs:element name="teachingMaterialID" type="xs:long" minOccurs="0"/>
<xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>

<!--
<xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
<xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
-->
</xs:sequence>
</xs:complexType>
</xs:schema>

Figure 43. ProblemsTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:include schemaLocation="ProblemToProblemSetAssociationsTable.xsd" />

<xs:element name="ProblemToProblemSetAssociations" type ="ProblemToProblemSetAssociations"/>
<xs:complexType name="ProblemToProblemSetAssociations">
<xs:sequence>
<xs:element ref="ProblemToProblemSetAssociationsTable" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

Figure 44. ProblemToProblemSetAssociations.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="ProblemToProblemSetAssociationsTable" type ="ProblemToProblemSetAssociationsTable"/>
<xs:complexType name="ProblemToProblemSetAssociationsTable">


<xs:sequence>
   <xs:element name="id" type="xs:long"/>
   <xs:element name="problemID" type="xs:long"/>
   <xs:element name="problemSetID" type="xs:long"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

Figure 45. ProblemToProblemSetAssociationsTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:include schemaLocation="ProblemTypesTable.xsd"/>

   <xs:element name="ProblemTypes" type="ProblemTypes"/>
   <xs:complexType name="ProblemTypes">
      <xs:sequence>
         <xs:element ref="ProblemTypesTable" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>

Figure 46. ProblemTypes.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="ProblemTypesTable" type="ProblemTypesTable"/>
   <xs:complexType name="ProblemTypesTable">
      <xs:sequence>
         <xs:element name="id" type="xs:long" minOccurs="0"/>
         <xs:element name="name" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>

Figure 47. ProblemTypesTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:include schemaLocation="QualityLevelsTable.xsd" />

<xs:element name="QualityLevels" type="QualityLevels"></xs:element>
<xs:complexType name="QualityLevels">
  <xs:sequence>
    <xs:element ref="QualityLevelsTable" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

Figure 48. QualityLevels.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema">

  <xs:element name="QualityLevelsTable" type="QualityLevelsTable"></xs:element>
  <xs:complexType name="QualityLevelsTable">
    <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="name" type="xs:string" minOccurs="0"/>
      <xs:element name="hide" type="xs:boolean" minOccurs="0"/>
      <xs:element name="restricted" type="xs:boolean" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

Figure 49. QualityLevelTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema">

<xs:include schemaLocation="ScaffoldsTable.xsd" />
<xs:include schemaLocation="Problems.xsd" />

<xs:element name="Scaffolds" type="Scaffolds"></xs:element>
<xs:complexType name="Scaffolds">
  <xs:sequence>
    <xs:element ref="ScaffoldsTable" minOccurs="0"/>
    <xs:element ref="Problems" maxOccurs="unbounded" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
Figure 50. Scaffolds.xsd

<x:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:x="http://www.w3.org/2001/XMLSchema">
  <xs:element name="ScaffoldsTable" type="ScaffoldsTable"/>
  <xs:complexType name="ScaffoldsTable">
    <xs:sequence>
      <xs:element name="id" type="xs:long" minOccurs="0"/>
      <xs:element name="name" type="xs:string" minOccurs="0"/>
      <xs:element name="tutorStrategyID" type="xs:long" minOccurs="0"/>
      <xs:element name="problemID" type="xs:long" minOccurs="0"/>
      <xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
      <xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

Figure 51. ScaffoldsTable.xsd

<x:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:x="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="TinyContentTable.xsd" />
  <xs:element name="TinyContent" type="TinyContent"/>
  <xs:complexType name="TinyContent">
    <xs:sequence>
      <xs:element ref="TinyContentTable" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

Figure 52. TinyContent.xsd

<x:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:x="http://www.w3.org/2001/XMLSchema">
<xs:element name="TinyContentTable" type="TinyContentTable"/>
<xs:complexType name="TinyContentTable">
  <xs:sequence>
    <xs:element name="id" type="xs:long" minOccurs="0"/>
    <xs:element name="externalReference" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

Figure 53. TinyContentTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="TutorStrategiesTable.xsd"/>
  <xs:include schemaLocation="TutorStrategyTypes.xsd"/>
  <xs:include schemaLocation="ContentExternalReferences.xsd"/>
  <xs:include schemaLocation="Hints.xsd"/>
  <xs:include schemaLocation="Scaffolds.xsd"/>

  <xs:element name="TutorStrategies" type="TutorStrategies"/>
  <xs:complexType name="TutorStrategies">
    <xs:sequence>
      <xs:element ref="TutorStrategiesTable" minOccurs="0"/>
      <xs:element ref="TutorStrategyTypes" minOccurs="0"/>
      <xs:element ref="ContentExternalReferences" minOccurs="0"/>
      <xs:element ref="Hints" maxOccurs="unbounded" minOccurs="0"/>
      <xs:element ref="Scaffolds" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

Figure 54. TutorStrategies.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema">
  <xs:element name="TutorStrategiesTable" type="TutorStrategiesTable"/>
  <xs:complexType name="TutorStrategiesTable">
    <xs:sequence>

<xs:element name="id" type="xs:long" minOccurs="0"/>
<xs:element name="contentExternalReferenceID" type="xs:long" minOccurs="0"/>
<xs:element name="name" type="xs:string" minOccurs="0"/>
<xs:element name="problemID" type="xs:long" minOccurs="0"/>
<xs:element name="tutorStrategyTypeID" type="xs:long" minOccurs="0"/>
<xs:element name="ownerUserID" type="xs:long" minOccurs="0"/>
<xs:element name="enabled" type="xs:boolean" minOccurs="0"/>
<xs:element name="createdAt" type="xs:dateTime" minOccurs="0"/>
<xs:element name="lastUpdated" type="xs:dateTime" minOccurs="0"/>
<xs:element name="deleted" type="xs:boolean" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>

Figure 55. TutorStrategiesTable.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="TutorStrategyTypesTable.xsd" />

  <xs:element name="TutorStrategyTypes" type ="TutorStrategyTypes">
    <xs:complexType name="TutorStrategyTypes">
      <xs:sequence>
        <xs:element ref="TutorStrategyTypesTable" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>

Figure 56. TutorStrategyTypes.xsd

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema"/>
Figure 57. TutorStrategyTypesTable.xsd

```xml
<xs:element name="TutorStrategyTypesTable" type = "TutorStrategyTypesTable"/>
<xs:complexType name = "TutorStrategyTypesTable">
  <xs:sequence>
    <xs:element name="id" type="xs:long" minOccurs="0"/>
    <xs:element name="name" type="xs:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Figure 58. Variables.xsd

```xml
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:include schemaLocation="VariablesTable.xsd" />

  <xs:element name="Variables" type = "Variables"/>
  <xs:complexType name = "Variables">
    <xs:sequence>
      <xs:element ref="VariablesTable" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```
<xs:element name="position" type="xs:long" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
Figure 59. VariableTable.xsd