Electronic Commerce Architecture Project
LAB Three: Writing Java Applets and GUI programs that Generate That Generate XML

by Lawrence Leff

In this lab, one creates a JAVA applet or GUI program with buttons and text fields. The user will be able to enter data and click buttons to create various parts of an XML document. You will learn how to display buttons and text fields in a program. You will learn how to make your program execute something when those buttons are pressed. Lastly, I explain the use of the Java "XML Serializer" classes which automatically write out the XML in an neat indented format, ready to be validated or sent to some other application.

These programs will use the JFC Swing library, which is the standard Graphical User Interface development environment from SUN, the originator of Java. We will also use the Xerces XML development library form IBM. This is a superset of the the Apache XML library. Apache is the most popular world web server software, beating out Windows NT and Netscape Server--it is also free.

We have some one-time activities to set up your account. This will involve downloading the Java Xerces parser material from the IBM directory and setting up a home page to which your program can write out XML. The S2wing Software is already in a MIT locker. The one-time activities include setting up the "add" commands to bring itin.

Then, I will discuss the program and its template so that you can extend it to other types of XML.

One-Time Operations

Download the IBM Xerces XML Java parser

1. Log into your athena account
2. Startup netscape and go to http://alphaworks.ibm.com/tech/xml4j
3. Click **download**
4. Select **XML4J-bin.3.0.1.tar.gz** which is the first file to download.
5. Click "**DownloadNow**"
6. You will see a license agreement. Enter your name, email address, and company. Select **I Accept This License Agreement**
7. Select **Download File**

8. Enter **/tmp/xml.tar.gz** in the **Selection** field and select **OK** The computer should start downloading.
9. In an xterm window, type `cd /tmp`
10. type `add sipb`
11. type `gunzip xml.tar.gz`
12. type `tar -xvf xml.tar`. The machine will extract many files to a directory. It will display a line as each file is extracted.
13. type `cd XML4J_3_0_1`
14. type `cp *.jar ~`
15. You may want to save the documentation and sample xml file directories. They aren't necessary for our work, however. So only do this if you have room in your account. If so type: `cp -r docs ~`
    and
    `cp -r data ~`
16. Create the file **adds** in your home directory containing:

```
17.  add -f java_v1.2.2
18.  add swing_v1.1
```

Each time you start work on this type of the program, type **source adds** to bring in the latest version of Java and swing, the GUI development environment. (Specifically, do this once for each xterm you start up.)

19. Create the file **c** in your home directory containing:

```
20.  #
21.  setenv SWING_HOME /mit/swing_v1.1/swing-1.1beta2
22.  setenv JAVA_DIR `attach -p java_v1.1.6`
23.  setenv JAVA_HOME ${JAVA_DIR}/distrib/${ATHENA_SYS}
24.  setenv CLASSPATH
   ".:xercesSamples.jar:xerces.jar:${SWING_HOME}/swingall.jar:
25.     ${SWING_HOME}/multi.jar:${JAVA_HOME}/lib/classes.zip"
26.  set path=(${JAVA_DIR}/arch/${ATHENA_SYS}/bin $path)
27.
28.  javac $1.java
```

Note, the two lines beginning "CLASSPATH" should be one line with no space between colon and ${SWING_HOME}.
Type `chmod 755 c`
c will be used to compile your Java Swing programs

29. Create the file **r** in your home directory containing:

```
30.  #
31.  setenv SWING_HOME /mit/swing_v1.1/swing-1.1beta2
```

```
32.  setenv JAVA_DIR `attach -p java_v1.2.2`
33.  setenv JAVA_HOME ${JAVA_DIR}/distrib/${ATHENA_SYS}
34.  setenv CLASSPATH
   .:xercesSamples.jar:xerces.jar:${SWING_HOME}/swingall.jar:
35.      ${SWING_HOME}/multi.jar:${JAVA_HOME}/lib/classes.zip
36.  java $1
```

Note, the two lines beginning "CLASSPATH" should be one line with no space between colon and ${SWING_HOME}.
Type `chmod 755 r`
The file **r** will be used to run your Java Swing programs.

Running a simple Java GUI program

Our first Java program creates a table of contents according the following lab1.dtd file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT tableofcontents (item*)>
<!ELEMENT item (Identifier,title)>
<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

To run it, be sure to do the following commands:

1. Type `source adds`. You should do this once, whenever you log in or startup a new xterm.
2. Down load the file N.java:
3. type `c N` (to compile the program)
4. type `r N` (to run the program)
   Several warnings will be written. These just tell you which version of the libraries used. You should ignore them.
5. Click the button labeled **Create Tableofcontents**
6. Enter information for the **Identifier** and **title** sub tags of your first **item** Click the button labeled **add item**.
7. Enter information for the **Identifier** and **title** sub tags of your second **item** Click the button labeled **add item**.
8. Repeat for any additional items, if desired.
9. Click the button **write it out** and dismiss the **N** dialogue.
10. Examine the **course.xml** file in the **www** directory. It should look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tableofcontents PUBLIC "lab1.dtd"
"lab1.dtd">
<tableofcontents>
<item>
```

```
<Identifier>w</Identifier>
<Title>w</Title>
</item>
</tableofcontents>
```

11. Validate your XML file as follows.

Template for a GUI Program that generates XML

The N.java program can be viewed as an instance of a general template. This template can be used whenever you have programs that allow the user to enter information to be put into an XML file. Most of the buttons will add information from the text boxes to part of the XML file being formed. It will either be a subtag for the root or main tag. Or it will be a deeper level tag that was created earlier from another button.

```
1: import com.sun.java.swing.*;

2: import java.awt.*;
3: import java.awt.event.*;

4: import org.apache.xerces.dom.*;
5: import org.apache.xml.serialize.*;
6: import org.w3c.dom.*;
7: import org.xml.sax.SAXException;
8: import java.io.*;
9: import java.lang.String;

10: public class N {
11: private FileWriter out;
12: private XMLSerializer X;
13: private Element root;
14: private DocumentImpl d;

15: public Component createComponents() {
16: // for each button
17:     JButton internal button name = new JButton("button title");
18:     button.addActionListener(new ActionListener() {
19:         public void actionPerformed(ActionEvent e) {
20:           try {
21:                     // put logic for button here
22:                     // there are templates below that you will need
23:        }
24:     });  // end of button declaration
25: //for each edit box or text field
26:        final JTextField text-nameField = new JTextField(number of
characters) ;
27:        JLabel text nameLabel = new JLabel ("caption for text field
");

28:        JPanel pane = new JPanel();
// Create a panel
```

```
29:         pane.setBorder(BorderFactory.createEmptyBorder(
// Grey background to border
30:                                             30, //top
31:                                             30, //left
32:                                             10, //bottom
33:                                             30) //right
34:                                             );
35:         pane.setLayout(new GridLayout(0, 1));
36:// one line for each button
37:         pane.add(internal button name);
38:         pane.add(text-nameLabel);
39:         pane.add(text-nameField);
40:         return pane;
41:     }

42:     public static void main(String[] args) {
43:         try {
44:             UIManager.setLookAndFeel(         // Be certain the
'look and feel' is supported
45:                 UIManager.getCrossPlatformLookAndFeelClassName());
46:         } catch (Exception e) { }

47:                                                 // Create the top-
level container and add contents to it.
48:         JFrame frame = new JFrame("title-bar for your application");
49:         N app = new N();
50:         Component contents = app.createComponents();
51:         frame.getContentPane().add(contents, BorderLayout.CENTER);

                                                  // Finish setting up the
frame, and show it.
52:         frame.addWindowListener(new WindowAdapter() {
53:             public void windowClosing(WindowEvent e) {
54:                 System.exit(0);
55:             }
56:         });
57:         frame.pack();
58:         frame.setVisible(true);
59:     }
60: }
```

Lines one through sixteen are boiler plate. Inside the createComponents() function, one creates each of the buttons, text fields and lables that one wishes to use. To create a button, one writes `JButton internal button name = new JButton("button title");` The *internal button name* is a name by which the button will be referred in the program. The *button title* is the name that the user will see on the button. Then, the command, `pane.add(internal button name);` (Line 37) physically puts the button on the screen.

Lines 21 to 22 show you where to put the Java logic for your button. A "listener" means that when you run the program, Java will wait for something to happen. In this case, it waits for the button to be pressed. Then, the indicated Java is executed.

Lines 26 show how one creates a place in which one can enter text. In our situation, this information will be read later and put into the XML:
`final JTextField text-nameField = new JTextField(number of characters) ;` In the button handling logic, the Java will read the information that the user typed into the place by writing `text-nameField.getText();`. And similarly to buttons, they are physically displayed on the window by writing: `pane.add(text-nameField);`

If one simply added a text field on the screen, the user would just see a white rectangle. They would have no idea what to type there, particularly, if there were many such rectangles on the screen. Thus, we add a lable just before each text box: `JLabel text nameLabel = new JLabel ("caption for text field ");` and `pane.add(text-nameLabel);`

Lines 28 to 35 create a pane which is a Java object which contains components such as edit boxes. In this simple example, we use **GridLayout (0,1)**. That means that all the components added will be in a vertical column. There are many other "layout managers" in Java Swing which allow one to have much more precise control of the look and arrangement of one's program. However, extensive discussions are beyond the scope of this course.

One button will have to start the creation of the XML file. The user will have to click this once at start. It will contain the following:

```
try {
  out = new FileWriter("www/filename.xml");
  out.flush();
  d = new DocumentImpl();
  root = d.createElement("name of root tag");
  d.insertBefore(root,null);
  OutputFormat o=new OutputFormat(d);
  o.setIndent(5);
  o.setIndenting(true);

  o.setDoctype("lab1.dtd","lab1.dtd");
  o.setDoctype("name of dtd file","name of dtd file");
  X = new XMLSerializer(o);
  X.setOutputCharStream(out);
  } catch (IOException e1){};
}
```

Another button will put to the disk the XML file. The user will click this when they have completed the operations to create the XML file. It will contain the following text as it appears below.

```
try {
   X.serialize(d);
   out.flush();
 } catch (IOException e2){};
```

All the other buttons will add information from one or more text fields to the edit boxes. You could also perform computations in them, such as totaling quantities.

For each XML tag that you wish to create, write:

```
Element tag-name;
tag-name = d.createElement("tag-name");
```

If the *tag-name* is to contain text from *field-name*, write:

```
String tag-nameString = field-nameField.getText();
TextImpl tag-nameText = (TextImpl)d.createTextNode
(tag-nameString);
tag-name.appendChild(tag-nameText);
```

If *tag-name* is to be put as a child of *other-tag*, one writes:

```
other-tag.appendChild(tag-name);
```

If *tag-name* goes just under the root, then one writes: `root.appendChild(tag-name);`

This is illustrated in the code for **button1** in **N.java** (reproduced below). This creates an **item** field. Then it creates two text fields for **Identifier** and **title**. The information for each of these is read from the text fields on the screen. The second and third from last lines show the **Identifier** tag and the **title** tag being added to the **item** tag we created in line two.

The last line adds the **item** we just created at the end of the list attached to the **tableofcontents**, our root element.

```
Element item;
item = d.createElement("item");
Element Identifier;
Identifier = d.createElement("Identifier");
String IdentifierString = IdentifierField.getText();
TextImpl IdentifierText = (TextImpl)d.createTextNode
(IdentifierString);
Identifier.appendChild(IdentifierText);

Element title;
title = d.createElement("title");
String titleString = TitleField.getText();
TextImpl titleText = (TextImpl)d.createTextNode
(titleString);
title.appendChild(titleText);
item.appendChild(Identifier);
item.appendChild(title);

root.appendChild(item);
```

Validating your XML

The **N.java** program you worked with will create a xml file in your **www** directory. You might wish to validate it. You can do this by downloading it to the **public_html** directory in your School of Architecture server account and following the instructions in Lab One.

The following procedure will work on the athena machines. You might find it more convenient.

Setting up your Home Page

You will find your validation work easiest if one puts your XML into a home page directory that is readable by the world. It will allow you to easily use validators such as those provided by: the Brown Scholarly Technology Group, World Wide WEB Consortium XML Schema Validation software, and the Microsoft validation page.
To do this on your Athena account, do these steps:

1. Log into your athena UNIX account
   If you are in the School of Architecture and Planning Computerized classroom, then use **Secure CRT** to log in. Enter **Start** and **Programs**, **Secure CRT 3.0** and **Secure CRT 3.0**. On the quick connect screen, enter **athena.dialup.mit.edu** in the host name field.
2. Type
   `mkdir ~/www`
3. Type
   `fs sa ~/www system:anyuser read`
4. Type
   `cd www`
5. create an html file called `home.html` and put some info there like `<P>Hello, I am` *`your name`*`.`
6. Type `chmod 755 home.html`.

MIT provides a good introduction to this procedure and pointers to resources on writing HTML.

Note: You only have to do the above four steps once! You will now have an established home page directory for the duration of your association with MIT.

You will need to refer to your home page as a URL from a web browser. Bring up Netscape or Internet Explorer. (The latter will be best in the School of Architecture Computerized Classroom)

Enter the following URL:
http://web.mit.edu/afs/athena.mit.edu/user/*n1*/*n2*/*user-id*/www/home.html
where *user-id* is your Athena user id, *n1* is the first letter of your Athena user id, *n2* is the second letter of your Athena user id. For example, my home page is:
http://web.mit.edu/afs/athena.mit.edu/user/l/e/leff/www/index.html
since my Athena user id is *leff*.

After a few days, you won't have to type such a tedious URL. You should be able to type http://web.mit.edu/*user-id*/www/home.html to get to your home page.

In general, if you create a file called *xxxx*.html in the **www** subdirectory of your account, you should be able to access it as
http://web.mit.edu/afs/athena.mit.edu/user/*n1*/*n2*/*user-id*/**www/***xxxx*.html
or
http://web.mit.edu/*user-id*/**www/***xxxx*.html to get to your home page.

Validation

1. Start up Netscape and go to the following URL (Bookmark it as you will be using it often.)
   http://www.stg.brown.edu/service/xmlvalid/
2. In the URI section of the form, enter the URL of the lab file that you just created:
   http://web.mit.edu/afs/athena.mit.edu/user/*n1*/*n2*/*user-id*/www/**lab1.xml**

Bibliography

Walrath, Kathy, and Mary Campione. *The JFC Swing Tutorial: A Guide to Constructing GUIs*. ISBN 0201433214. Addison Wesley, 1999.

Xerces XML Parser Documentation. http://alphaworks.ibm.com/tech/xml4j