Project 3

Wahoo!

124 Points

due by 12:30 P.M. ET on Wednesday, 4 April 2007

It is recommended that you read the entirety of this document,
well in advance of this project's deadline, before answering any of its questions.

It is further recommend that you begin answering this document's questions immediately thereafter.
Really.

Goals.

The goals of this project are to:

- Challenge you to implement your own XML-based portal.
- Give you hands-on experience with JavaServer Pages 2.1, Java Servlet 2.5, and TrAX, as well as additional experience with XSLT.
- Introduce you to an industry-standard application server, Apache's Tomcat 6.0.10.

Grading Metric.

Each question is worth the number of points specified parenthetically in line with it.

Your responses to questions requiring exposition will be graded on the basis of their clarity and correctness. Your responses to questions requiring code will be graded on the following bases.[1]

Basis	Considerations
Correctness	Does your code work in accordance with the question's guidelines?
Design	Does your code make sense, given the question's framework? Is your code written logically, clearly, and succinctly? Is your code efficient? Is your code divided into logical units (*e.g.*, multiple methods and templates)?
Style	Is your code rigorously documented with inline comments and Javadoc doc comments and tags?[2] Is it clear from your comments alone how your code operates? Is your code pretty-printed? Are your attributes, data members, elements, methods, templates, and variables aptly named?

Academic Honesty.

For this project, you will be running your own instance of a webserver on `nice.fas.harvard.edu` that does not require that any files or directories being served be world-executable or world-readable.

If, however, you opt to publish world-readable files on `www.people.fas.harvard.edu` for some reason, it is expected that you will ensure the privacy of your work by password-protecting any Web-accessible directories and naming files in such a (random) way as to render their viewing on `nice.fas.harvard.edu` unlikely.

Recall that, for Project 1, you configured your account with a `~/public_html/` directory and a password-protected `~/public_html/cscie259/` directory.

Needless to say, attempting to view the work of another student, even if published in a world-accessible directory, is considered academic dishonesty and will be handled accordingly.

[1] By code, we mean Java, JSP, XHTML, XML, XPath, and XSLT.
[2] With regard to Javadoc, we expect descriptions for all methods and appropriate use of the `@param`, `@return`, and `@throws` block tags.

Getting Started.

1. (1 point.) Okay, okay, fine, here's a point.

2. (0 points.) If you intend to do your work on `nice.fas.harvard.edu`, SSH to that machine and execute the following sequence of commands.[3,4]

```
cp -r ~cscie259/pub/distribution/projects/project3-7.0/ ~/cscie259/
cd ~/cscie259/
ls
```

You should see that you have the following in your current working directory.

```
project1-7.0/       project2-7.0/     project3-7.0/
```

If, on the other hand, you do not intend to do your work on `nice.fas.harvard.edu`, you may proceed to download a `gzip`-compressed tarball or a ZIP file containing this `project3-7.0/` directory from the course's website to your local machine. Or, of course, you can transfer the directory itself via SFTP to your local machine. Refer to this document's Appendix for further directions.

Notice, now, that your `project3-7.0/` directory is structured as follows.

```
project3-7.0/
  conf/
  src/
   cscie259/
     project3/
       wahoo/
  temp/
  webapps/
    ROOT/
      docs/
        cscie259/
          project3/
            wahoo/
      dtd/
      images/
      WEB-INF/
      xml/
        cache/
      xsl/
```

Ensure that your account or machine is in order by executing the following command from within any directory.[5]

[3] These commands assume that you already created (for Projects 1 and 2) a directory called `cscie259/` in your FAS account's home directory.
[4] Beware the distinction between `~cscie259` and `~/cscie259`.
[5] Note that, on `nice.fas.harvard.edu`, `` `EnvironmentCheck` `` is an alias for
`` `java org.apache.xalan.xslt.EnvironmentCheck` ``.

```
EnvironmentCheck
```

Confirm that your account is configured to use Ant 1.7.0, Xerces-J 2.7.1, and Xalan-J 2.7.0.

Next, execute the following command from within your `project3-7.0/` directory.[6]

```
ant compile
```

By default, `ant` will compile `cscie259.project3.wahoo.*`, storing the resulting bytecodes in `project3-7.0/webapps/ROOT/WEB-INF/classes/`.

In fact, take a look at `project3-7.0/build.xml` so that you know what `ant` can do for you during this project.

Next, proceed to edit `project3-7.0/conf/server.xml` with your favorite text editor. Per the file's comments, locate the `Server` element's `port` attribute and assign it an integral value between 1024 and 65535, inclusive; then, locate the `Connector` element's `port` attribute and assign it an integral value between 1024 and 65535, inclusive, as well, taking care not to reuse the value you chose for the `Server` element's `port` attribute. After saving and closing the file, execute the following command from within `project3-7.0/`.[7]

```
tomcat
```

Then, assuming you've connected via SSH to `nice.fas.harvard.edu`, take notice of the particular host to which you're connected by examining your prompt (which should be of the form `"$USER@$ICEBOX (%~):"`, per Project 1's configuration of your account) or by executing the following command.[8]

```
echo $ICEBOX
```

You should see output resembling the below, where `n` is your choice of ports for the `Connector` element's `port` attribute.

[6] Alternatively, you can execute `ant` without any arguments.

[7] Note that, on `nice.fas.harvard.edu`, `` `tomcat` `` is an alias for `` `catalina.sh run` ``. If you opt to develop on a Windows machine, you should instead execute `` `catalina.bat run` `` (assuming `"%CATALINA_HOME%\bin\"` is in your `PATH`); alternatively, you can achieve this same aliasing effect by creating in any directory in your `PATH` a file called `tomcat.bat` containing the following two lines.

```
@echo off
%CATALINA_HOME%\bin\catalina.bat run
```

[8] Though we've always referred to it as one system, `nice.fas.harvard.edu` actually refers to a cluster of machines, all of which mount via NFS your home directory; you are connected to one essentially at random.

```
Using CATALINA_BASE:   .
Using CATALINA_HOME:   /home/c/s/cscie259/pub/local/jvm/apache-tomcat-6.0.10
Using CATALINA_TMPDIR: ./temp
Using JRE_HOME:       /home/c/s/cscie259/pub/local/i386/jdk1.5.0_11
Mar 11, 2007 10:54:00 PM org.apache.coyote.http11.Http11Protocol init
INFO: Initializing Coyote HTTP/1.1 on http-n
Mar 11, 2007 10:54:00 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 838 ms
Mar 11, 2007 10:54:00 PM org.apache.catalina.core.StandardService start
INFO: Starting service Catalina
Mar 11, 2007 10:54:00 PM org.apache.catalina.core.StandardEngine start
INFO: Starting Servlet Engine: Apache Tomcat/6.0.10
Mar 11, 2007 10:54:00 PM org.apache.coyote.http11.Http11Protocol start
INFO: Starting Coyote HTTP/1.1 on http-n
Mar 11, 2007 10:54:00 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 664 ms
```

At this point, an instance of Tomcat is running under your userid on port `n` of the host to which you're connected. (Tomcat should now have control of your terminal; you should not be returned to your shell's prompt.) Confirm as much by visiting the URL below, taking care to substitute `$ICEBOX` and `n` with their appropriate values.[9,10]

```
http://$ICEBOX.fas.harvard.edu:n/happyenv.jsp
```

Confirm that Tomcat, like your account, is configured to use Xalan-J 2.7.0 and Xerces-J 2.7.1.

Next, visit the URL below, again taking care to substitute `$ICEBOX` and `n` with their appropriate values.

```
http://$ICEBOX.fas.harvard.edu:n/
```

You should be automatically directed to `/servlet/login`, which should display Wahoo's login page. Meanwhile, output resembling the below should have been outputted to Tomcat's console (*i.e.*, your terminal).

```
Adding "jharvard" to main memory with password "crimson"...  Done.
Associating "Boston news" with user "jharvard"...  Done.
Associating "Top stories" with user "jharvard"...  Done.
Associating "Top technology stories" with user "jharvard"...  Done.
```

Go ahead and log in with username `"jharvard"` and password `"crimson"`. Though, seeing as you haven't yet implemented your portal, the results won't be all that interesting.

Proceed to terminate Tomcat by hitting `Ctrl-C` at its console (*i.e.*, your terminal).

If your account or system does not behave as described above, simply contact `cscie259@lists.dce.harvard.edu` or the staff for assistance.

[9] If you opt to develop on your own machine, you should instead visit `http://localhost:n/` or `http://127.0.0.1:n/`.

[10] Although you needn't type "`http://`" to visit most websites with most browsers these days, you might need to do so in this case.

My Wahoo!

3. (0 points.) "Moreover Technologies is a provider of real-time information management solutions, delivering essential online information in time to impact business decisions.

   "Unlike traditional news services that resell archived information, Moreover's sophisticated technology continually scours the Internet to capture breaking news and business information from more than 4,500 qualified, handpicked sources. Headline links to relevant information are filtered according to clients' specific business needs, and delivered in real time to any platform or business application through one of Moreover's customizable Connected Intelligence™ solutions.

   "Corporations use Moreover's technology solutions to stay competitive and increase their margins. Today's quickly evolving business environment requires that companies have the right information, all the time, to spot market opportunities, track industry developments, and react quickly to announcements. Ultimately, Moreover helps every employee make better and faster decisions to stay ahead of the competition.

   "Moreover Technologies has operations in San Francisco and London, and is backed by prominent venture firms including Reuters Venture Capital and Atlas Ventures."[11]

   How's that for marketing hype?

   In other words, Moreover is a syndicator of news. The firm aggregates headlines from around the world and produces "over 330 feeds that are free to non-for-profit webmasters," who can then incorporate those headlines and the associated articles into their own sites.[12]

   Sound too good to be true? Take a look for yourself at Moreover's offerings by visiting the URL below!

   `http://w.moreover.com/categories/category_list.html`

   Clearly, Moreover has incorporated their headlines into a portal of their own. But it's no Wahoo! That's where you come in. Read on.

4. (0 points.) Not only does Moreover offer its newsfeeds as HTML, it also offers them as XML, in both RSS and its own proprietary format. Needless to say, we'll be using XML, Moreover's "preferred format [for] developers' use for integration."[13] Specifically, we'll be using Moreover's proprietary format, only because it offers more details than does its RSS feed (*e.g.*, articles' harvest times).

[11] Quotation excerpted from `http://w.moreover.com/main_site/aboutus/company.html` prior to page's alteration.
[12] Quotation excerpted from `http://w.moreover.com/dev/xml/` prior to page's removal.
[13] Quotation excerpted from `http://w.moreover.com/dev/` prior to page's alteration.

Before you can access a newsfeed, however, you must know its name. Fortunately, Moreover provides an XML-based list of "channels" and categories into which it has placed its many newsfeeds, the DTD for which appears below.[14]

```
<!ELEMENT nested_category_list (channel*)>
<!ELEMENT channel (channel_name, category*)>
<!ELEMENT channel_name (#PCDATA)>
<!ELEMENT category (category_name, feed_name)>
<!ELEMENT category_name (#PCDATA)>
<!ELEMENT feed_name (#PCDATA)>
```

The list of channels and their categories itself is available (as, of course, XML) at the URL below.[15]

```
http://w.moreover.com/categories/nested_category_list.xml
```

Note that each newsfeed mentioned in the list of channels and categories can be accessed by way of its `category_name`. For instance, to access the "Biotech news" feed, you can visit

```
http://p.moreover.com/cgi-local/page?c=Biotech%20news&o=xml
```

where the HTTP GET string's parameters specify the category's name (`c`) and the newsfeed's output format (`o`), respectively.[16]

Each feed is then structured according to the DTD below.[17]

```
<!ELEMENT moreovernews (article*)>
<!ELEMENT article (url, headline_text, source, media_type, cluster,
tagline, document_url, harvest_time, access_registration, access_status)>
<!ATTLIST article id ID #IMPLIED>
<!ELEMENT url (#PCDATA)>
<!ELEMENT headline_text (#PCDATA)>
<!ELEMENT source (#PCDATA)>
<!ELEMENT media_type (#PCDATA)>
<!ELEMENT cluster (#PCDATA)>
<!ELEMENT tagline (#PCDATA)>
<!ELEMENT document_url (#PCDATA)>
<!ELEMENT harvest_time (#PCDATA)>
<!ELEMENT access_registration (#PCDATA)>
<!ELEMENT access_status (#PCDATA)>
```

For instance, below is an excerpt from the "Biotech news" feed at 10:21 P.M. ET on Sunday, 11 March 2007.[18]

---

[14] This DTD is also available at `http://w.moreover.com/xml/xml_nestedcatlist.dtd` as well as in `project3-7.0/webapps/ROOT/xml/cache/`.

[15] A sample list, generated automatically at 10:25 P.M. ET on Sunday, 11 March 2007, is available in `project3-7.0/webapps/ROOT/xml/cache/nested_category_list.xml`.

[16] Note that any space in a category's name must be encoded in the `application/x-www-form-urlencoded` MIME format (*i.e.*, as `'+'` or `'%20'`).

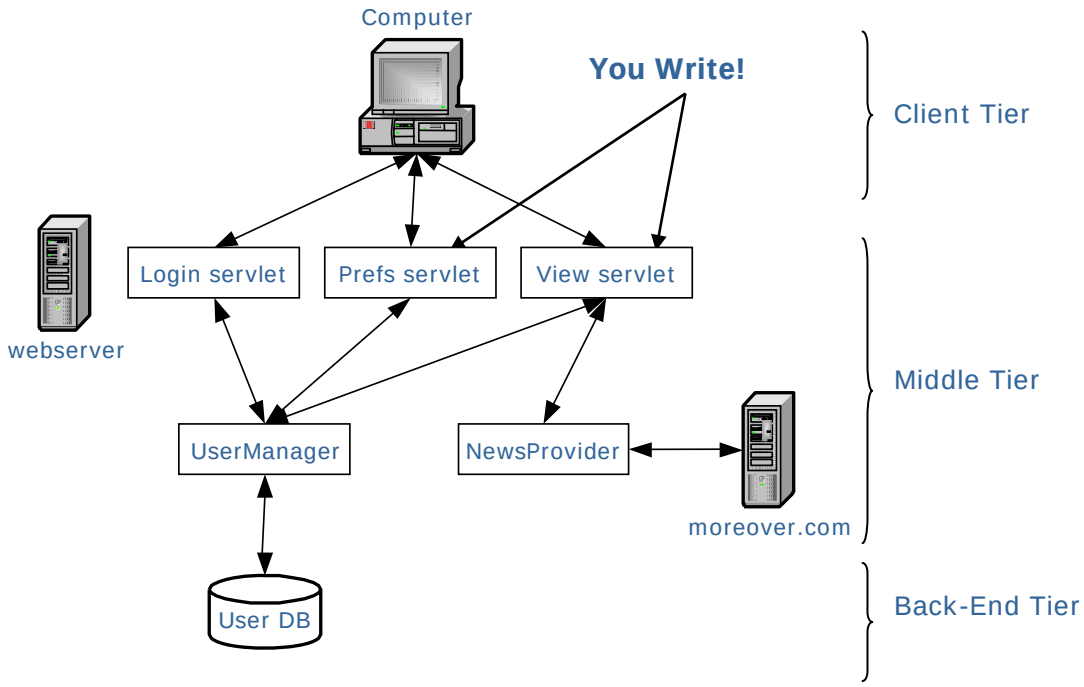
[17] This DTD is also available at `http://p.moreover.com/xml_dtds/moreovernews.dtd`.

```
<moreovernews>
  [...]
  <article id="_840925179">
    <url>http://c.moreover.com/click/here.pl?x840925179</url>
    <headline_text>Whose Genome Is It, Anyway?</headline_text>
    <source>Discover</source>
    <media_type>text</media_type>
    <cluster>moreover...</cluster>
    <tagline></tagline>
    <document_url>http://discovermagazine.com</document_url>
    <harvest_time>Mar 11 2007 8:46AM</harvest_time>
    <access_registration></access_registration>
    <access_status></access_status>
  </article>
  [...]
</moreovernews>
```

Clearly, the root element of the newsfeed is `moreovernews`. Meanwhile, an `article` element represents an individual article, uniquely identified by way of an `id` attribute. Each `article` has a number of children: `url` specifies a proxy address for the actual article; `headline_text` specifies the article's headline; `source` specifies the article's source; `media_type` specifies the article's format; `cluster` specifies (so far as I can tell) some sort of grouping for the article; `tagline` specifies a the article's tagline, if any; `document_url` specifies the homepage of the article's source; `harvest_time` specifies the moment at which the headline was discovered; and `access_registration` and `access_status` specify registration requirements for the article's source, if any.

5. (0 points.) Lest you be overwhelmed by the challenge of implementing a web-based portal, rest assured that we've provided you with a framework. Per Lecture 7, Wahoo's architecture resembles the below.

[18] A copy of this excerpt's source is available in `project3-7.0/webapps/ROOT/xml/cache/Biotech%20news.xml`.



In order to better understand this architecture, spend some time looking through the code in `project3-7.0/src/cscie259/project3/wahoo/`.

Start by examining the `Login` servlet by way of `Login.java`, which handles the task of authenticating and creating users; notice how it uses `login.xsl` in `project3-7.0/webapps/ROOT/xsl/` to generate the login screen. Notice how HTTP GETs and HTTP POSTs are both handled by `doWork(·,·,·)`, which essentially drives the entire servlet. Since Wahoo's login screen is rather minimalist, `displayLoginForm(·,·,·)` does little more than apply (by way of TrAX) `login.xsl` to an empty document; certainly a more interesting login screen could make use of a non-empty document or actual XML content. Note, by the way, that the XHTML generated by `login.xsl` includes a `form` that will be submitted via an HTTP POST operation back to the `Login` servlet. Finally, notice how the mere presence of a username in the session implies that a user is currently authenticated (useful knowledge if you wish to detect whether a user's attempted to visit some servlet prior to authenticating).

Take a quick peek, by the way, at `project3-7.0/webapps/ROOT/index.jsp`, a minimalist JSP that redirects users to the `Login` servlet. In other words, if a user visits `/` on your instance of Tomcat, the user will be redirected automatically to `/servlet/login`.

Next, take a look at the `View` servlet by way of `View.java`; recall from your perusal of `Login.java` that the `Login` server redirects a user to this `View` servlet upon successful authentication. Eventually, this `View` servlet will generate the user's personalized portal; at present, it does very little.

Now, stroll on over to the `Prefs` servlet by way of `Prefs.java`. Whereas the `View` servlet will eventually display a user's favorite headlines, the `Prefs` servlet will eventually provide the user with a XHTML form-based interface with which to add newsfeeds to and remove newsfeeds from his favorites. At present, the `Prefs` servlet does very little.

That's it for Wahoo's servlets. To understand how they're accessed by way of `/servlet/`, take a quick look at `project3-7.0/webapps/ROOT/WEB-INF/web.xml`.

Okay, let's now examine the remaining classes in `project3-7.0/src/cscie259/project3/wahoo/`, all of which provide support for Wahoo's middle tier, but don't interact with the client tier directly.

First, take a gander at `WahooServlet.java` for `WahooServlet`, the abstract class from which the `Login`, `Prefs`, and `View` servlets descend. Besides the signature for each of those classes' `doWork` methods, `WahooServlet` also offers a pair of helper methods, one of which (*i.e.*, `redirect(·,·)`) the `Login` servlet already makes use of, the other of which (*i.e.*, `forward(·,·,·)`) you may or may not find useful during development.

Next, steal a glance at `User.java`, which implements the concept of a user. Then, read over `UserManager.java`, which handles the storage and retrieval of user data, including usernames, passwords, and favorite newsfeeds. In fact, recall the output on Tomcat's console when you tried to log in earlier: it's this `UserManager` class whose methods loaded John Harvard and his favorites into memory. Not sure how such came to pass? Well, upon the user's first visit to the `Login` servlet, the servlet container automatically executes that servlet's `init()` method, inherited from `WahooServlet`. (Similarly does the servlet container handle the user's first visit to the `Prefs` servlet.) That `init()` method, meanwhile, invokes the `UserManager` class's own `init(·)` method, which proceeds to instantiate, if it hasn't already, the class's one and only instance, the result of which is execution of the class's `private`, explicit constructor, which parses `project3-7.0/webapps/ROOT/xml/users.xml` and loads John Harvard and his favorites into memory. Make sense?[19] Take a look at `users.xml`, so that you see exactly what data is loaded into memory.

Some additional discussion of the `User` and `UserManager` classes is perhaps in order. First, it's worth noting that calling, for example,

```
UserManager.getUser("jharvard")
```

will return John Harvard's `User` object, provided John is registered in `users.xml` (which he is, by default); otherwise, this method returns `null`.

[19] Feel free to insert temporarily, for example,

```
(new Exception()).printStackTrace()
```

anywhere you'd like to force a printing of the call stack; such should make this sequence of executions all the more clear.

Second, calling, for instance,

```
UserManager.addUser("jharvard", "crimson")
```

will add to Wahoo's database a user with username `"jharvard"` and password `"crimson"`, thereafter returning a reference to the newly instantiated `User` object, unless `username` `"jharvard"` is already in the database, in which case `null` is returned.

Finally, calling

```
UserManager.save()
```

will save information about users to `users.xml`. Be sure to call this method every time you change user information (*e.g.*, add a user, add or remove a favorite newsfeed for a user, *etc.*), else the information will be lost when Tomcat terminates.

Note that each of these three methods is static; you will never manipulate a reference to `UserManager` object directly. Additionally, you should never instantiate a `User` object directly; allow the `UserManager` class's `addUser(·,·)` method to handle such details.[20]

Make sense? Okay, last but not least, look over `NewsProvider.java`, which handles the retrieval of headlines from Moreover.

If you learn better by way of Javadoc, feel free to view `project3-7.0/webapps/ROOT/docs/` with your favorite browser. Note that we've placed the project's initial Javadoc in `project3-7.0/webapps/ROOT/` rather than in `project3-7.0/` so that you can view our (and, soon, your) Javadoc with your own instance of Tomcat![21] In fact, if Tomcat is running, you can visit the URL below, again taking care to substitute `$ICEBOX` and `n` with their appropriate values.

```
http://$ICEBOX.fas.harvard.edu:n/docs/
```

Got it? Lots of code, to be sure. But, trust us: spend a good amount of time know getting to know the distribution code. Only if you understand the framework will building upon it be straightforward.

6. (1 point.) You're doing well, so far. Here's a point for hanging in there!

7. (40 points.) Okay, here we go! Your first challenge is to implement the essence of the portal: the `View` servlet, which, once a user has authenticated, displays, at least, the user's favorite headlines and links to the corresponding articles. The design of this page (or, rather, set of

[20] In the interests of testing, you're welcome, of course, to add `user` elements to `users.xml` with your favorite text editor.

[21] Know that, during the development of this project, you can update your Javadoc by executing `` `ant javadoc` `` from within your `project3-7.0/` directory.

pages, all of which will be generated dynamically) is largely up to you, but we do ask that you do adhere to a few guidelines.

i. The servlet should allow a user to view headlines from any of his or her favorite newsfeeds. For instance, by clicking a link labeled `"Boston news"`, John Harvard should be able to view Boston's latest headlines (along with links to the corresponding articles); that link should pass back to the servlet the user's choice of newsfeeds (*i.e.*, categories) by way of an HTTP parameter. In other words, the servlet should *not* output the headlines from all of John Harvard's favorites newsfeeds in one, large page. We suggest that you include at the bottom of any page generated by the servlet a list of the user's favorite newsfeeds, so that navigation among their headlines is simple.

ii. The servlet should utilize `project3-7.0/webapps/ROOT/xsl/view.xsl` in order to generate these pages of headlines dynamically, making use, of course, of the XML returned by the `NewsProvider` class's `getHeadlines(·)` method.

iii. If a user attempts to visit the servlet prior to authenticating, you should redirect the user to the `Login` servlet.

iv. Any page generated by the servlet should contain a link to the `Prefs` servlet, which, of course, will be implemented by you shortly!

v. Any page generated by the servlet should contain a link via which the user can log out (*e.g.*, the link might pass some parameter back to the servlet, instructing it to invalidate the current session and redirect the user to the `Login` servlet).

Most likely, you will want to pass to `view.xsl` not only the XML for some newsfeed but also the names of the user's other favorite categories (in fulfillment of i, above). Clearly, you can obtain the former by way of the `NewsProvider` class's `getHeadlines(·)` method. The latter you can obtain by way of a `User` object's `getNewsCategories()` method. However, those categories are returned as an instance of `java.util.List`, not XML. Hence, you may want your `View` servlet to iterate through this `List` of categories, wrapping each category with some XML start tag and corresponding end tag, concatenating the elements together to form a `String`. You can then wrap those elements with a parent element (by concatenating the `String` with some other start tag and end tag on either side). You can then concatenate that result with the XML returned by `getHeadlines(·)` (the root of which is a `moreovernews` element), thereafter encapsulating the data in a single, root element. You can then apply `view.xsl` to that dynamically created `String`.[22]

However, since the XML is in a `String`, you cannot utilize TrAX in the same manner that the `Login` servlet does. Recall that the `Login` servlet applies `login.xsl` to `dummy.xml` by way of the following code.

[22] Realize that this `String`'s dynamic generation is inelegant (if not an abuse of XML); we recommend it here only so that you can focus on other aspects of the project.

```
StreamSource xmlSource =
new StreamSource(getServletContext().getRealPath("/xml/dummy.xml"));
transformer.transform(xmlSource, new StreamResult(out));
```

To apply `view.xsl` to a `String` called, say, `foo` (as opposed to a file), you'll need to employ code like the below (provided your `Transformer` object has already been configured with `view.xsl`, just as `Login.java`'s `Transformer` object is configured with `login.xsl`).

```
StreamSource xmlSource =
 new StreamSource(new java.io.ByteArrayInputStream(foo.getBytes()));
transformer.transform(xmlSource, new StreamResult(out));
```

Incidentally, although we've configured Tomcat to check the modification dates and times of class files, you may want to restart Tomcat anytime you recompile your source.

Okay, get to it!

8. (40 points.) Now that your `View` servlet is working, it's time to enable the `Prefs` servlet, so that the user can customize his or her favorite categories. While the design of this interface is also largely up to you, we do ask that you do adhere to a few guidelines.

   i. The servlet should enable the user to add newsfeeds to his or her favorites as well as remove newsfeeds from the same. The underlying mechanism for doing so should be an XHTML `form`, which should be submitted back to your `Prefs` servlet by way of HTTP GET or POST. You may find that `checkboxes` are the most aesthetically pleasing choice of input mechanisms. However, the servlet should *not* output a list of all available newsfeeds in one, large page. Instead, we suggest that you display only one channel of categories at a time, allowing the user to navigate among the available channels by way of hyperlinks or forms, which submit some parameter that instructs your `Prefs` servlet to display a different channel.

   ii. The servlet should utilize `project3-7.0/webapps/ROOT/xsl/prefs.xsl` in order to generate its pages dynamically.

   iii. If a user attempts to visit the servlet prior to authenticating, you should redirect the user to the `Login` servlet.
   iv. Any page generated by the servlet should contain a link to the `View` servlet.
   v. Any page generated by the servlet should contain a link via which the user can log out (*i.e.*, the link will pass some parameter back to the servlet, instructing it to invalidate the current session and redirect the user to the `Login` servlet).

   Again, be sure to call the `UserManager` class's `save()` method anytime a user updates his or her preferences.

   As in the `View` servlet, you may find it useful (if not necessary) to concatenate an XML representation of the user's current favorites with an XML representation of all available

categories so that the servlet is able to report simultaneously (by way of `prefs.xsl`) the user's current favorites as well as other, available categories (in, say, some channel).

Again, although we've configured Tomcat to check the modification dates and times of class files, you may want to restart Tomcat anytime you recompile your source.

Alright, make it happen!

9. (40 points.) Pretty neat stuff, eh? By now, you're hopefully feeling pretty good about your application. And you should; you've pulled together a lot of material, a lot of skills.

   It's time, then, to make that portal truly yours. Although we claim to have left the design of your portal largely up to you, it was certainly our idea to have a login page, a viewing page, and a preferences page. What functionality did we leave out, though? What would make you want to keep that instance of Tomcat running indefinitely, your portal available 24/7?

   Go ahead and add a personal touch to your portal. It should go without saying that you're welcome to beautify your portal with colors and images. But we'd like you to extend your portal's functionality. The extension could be in the client tier. What's lacking in your portal's user interface? The extension could also be in the middle tier. What feature are your servlets missing? What servlet is missing? Or, the extension could be in the back-end tier. Right now, your code (presumably) relies upon the copy of `nested_category_list.html` in `project3-7.0/webapps/ROOT/xml/cache/`. What if Moreover's offerings have been updated since we cached that file for you? Perhaps your portal should be retrieving a fresh listing on occasion, re-caching the results. In fact, for the sake of performance, perhaps the `NewsProvider` class should be caching headlines, at least for a short while, so that it needn't open up a TCP socket for every request! Also, what about those DTDs? Should you be validating your XML feeds? Why not spend time on all three tiers? Isn't RSS all the rage these days? Why not support feeds from other sources?

   In short, make your portal better. Feel free to discuss (but not collaborate on the implementation of) possible extensions with other students by way of `cscie259@lists.dce.harvard.edu`. You may certainly incorporate additional technologies (*e.g.*, JSP, XSL-FO, *etc.*) into your extension. We ask only that the scope of your extension be comparable to that required by question 7 or 8; in other words, the extension should demand of you about the same amount of time as did your `Login` servlet or `Prefs` servlet. You are welcome (and encouraged) to run a possible extension by the staff if you're worried that it may fall short of our expectations.

   Once you've designed and implemented your extension, create in `project3-7.0/` a file called `personaltouch.{html,pdf,ps,rtf,txt}` containing two or more paragraphs detailing your extension. Be sure to explain what your extension is, how and in what file(s) you implemented it, and what challenges you encountered during its implementation.

10. (2 points.) Wahoo! You're done. Two points!

Submitting Project 3.

11. (0 points.) If you have not done your work on `nice.fas.harvard.edu`, transfer via SFTP your `project3-7.0/` directory from your local machine to the `cscie259/` directory in your FAS account's home directory, taking care to upload any text files in ASCII mode.

    Next, ensure that the structure of your `project3-7.0/` directory on `nice.fas.harvard.edu` is the following.

```
project3-7.0/
  conf/
  src/
   cscie259/
     project3/
       wahoo/
  temp/
  webapps/
    ROOT/
      docs/
        cscie259/
          project3/
            wahoo/
      dtd/
      images/
      WEB-INF/
      xml/
        cache/
      xsl/
```

    Your `project3-7.0/` directory should contain `build.xml` (updated, if necessary, to handle any files of your own creation) and `personaltouch.{html,pdf,ps,rtf,txt}`; `project3-7.0/conf/` should contain your application's configuration files; `project3-7.0/webapps/ROOT/docs/` should contain updated Javadoc; `project3-7.0/webapps/ROOT/dtd/` should contain those DTDs it originally contained, along with any others you might have created; `project3-7.0/webapps/ROOT/images/` should contain the image it originally contained, along with any others you might have created; `project3-7.0/webapps/ROOT/WEB-INF/` should contain `web.xml`; `project3-7.0/src/` should contain the Java source it originally contained, along with any additional Java source you wrote; `project3-7.0/webapps/ROOT/xml/` should contain those XML files it originally contained, along with any others you might have created; and `project3-7.0/webapps/ROOT/xsl/` should contain those XSLT files it originally contained, along with any others you might have created.

    Particularly if you developed on another machine, ensure that your application builds and executes properly on `nice.fas.harvard.edu`.

    Once everything's in order, clean up your workspace by executing

```
ant clean
```

from within your `project3-7.0/` directory.[23]

Finally, submit your work electronically by executing the following command from within your `project3-7.0/` directory.

```
cscie259submit project3
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your project's successful submission. You may re-submit as many times as you'd like; each re-submission will overwrite any previous submission. But take care not to re-submit after the project's deadline, as only your latest submission's timestamp is retained.

[23] Know that this command deletes not only your bytecodes but also Tomcat's `logs`, runtime directories, and temporary files.

Appendix

So long as your project ultimately compiles and executes on `nice.fas.harvard.edu`, you are welcome to develop it on your own computer. We leave it to you to translate the commands in this document to your own operating system's syntax (*e.g.*, forward slashes to backslashes, `$VAR` to `%VAR%`, *etc.*). This appendix explains how to configure your computer like `nice.fas.harvard.edu`.

You are encouraged to discuss and troubleshoot these steps with classmates via the listserv. Some of these steps also appeared in the specifications for Project 1 (①) and/or Project 2 (②).

⊗ Get ready to start crossing off circles.

① Download JDK 5.0 Update 11 via the course's website and install it.[24]
① Define an environment variable called `JAVA_HOME` whose value is the full path to the JDK's directory.
① Prepend `"$JAVA_HOME/bin"` to your `PATH`.
① Define an environment variable called `JAVA_COMPILER` whose value is `"NONE"`.
○ Create a directory in `$JAVA_HOME/jre/lib/` called `endorsed`.

① Prepend `"./build"` to your `CLASSPATH`.
① Prepend `"."` to your `CLASSPATH`.

① Download Xalan 2.7.0 (*i.e.*, `xalan-j_2_7_0-bin.{tar.gz,zip}`) via the course's website and extract it to its own directory.
① Define an environment variable called `XALAN_HOME` whose value is the full path to Xalan's directory.
○ Remove `"$XALAN_HOME/serializer.jar"` from your `CLASSPATH`.
○ Place a copy of `$XALAN_HOME/serializer.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
○ Remove `"$XALAN_HOME/xalan.jar"` from your `CLASSPATH`.
○ Place a copy of `$XALAN_HOME/xalan.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
○ Remove `"$XALAN_HOME/xercesImpl.jar"` from your `CLASSPATH`.
○ Place a copy of `$XALAN_HOME/xercesImpl.jar` in `$JAVA_HOME/jre/lib/endorsed/`.
○ Remove `"$XALAN_HOME/xml-apis.jar"` from your `CLASSPATH`.
○ Place a copy of `$XALAN_HOME/xml-apis.jar` in `$JAVA_HOME/jre/lib/endorsed/`.

[24] Sun's installer for Windows installs both a JDK and a JRE, the latter of which effectively has higher priority in the operating system's `PATH`; Windows users may wish to uninstall this JRE via Add or Remove Programs in their Control Panel.

② Define an alias called `EnvironmentCheck` whose value is `"java org.apache.xalan.xslt.EnvironmentCheck"`.[25]
② Define an alias called `xalan` whose value is `"java org.apache.xalan.xslt.Process"`.[26]

① Download Ant 1.7.0 via the course's website and extract it to its own directory.
① Define an environment variable called `ANT_HOME` whose value is the full path to Ant's directory.
① Add `"$ANT_HOME/lib/ant.jar"` to your `CLASSPATH`.
① Add `"$ANT_HOME/lib/ant-launcher.jar"` to your `CLASSPATH`.
① Append `"$ANT_HOME/bin"` to your `PATH`.

○ Download Tomcat 6.0.10 (*i.e.*, `apache-tomcat-6.0.10.{tar.gz,zip}`) via the course's website and extract it to its own directory.
○ Define an environment variable called `CATALINA_HOME` whose value is the full path to Tomcat's directory.
○ Define an environment variable called `CATALINA_BASE` whose value is `"."`.
○ Add `"$CATALINA_HOME/bin/bootstrap.jar"` to your `CLASSPATH`.
○ Add `"$CATALINA_HOME/lib/servlet-api.jar"` to your `CLASSPATH`.
○ Create a directory in `$CATALINA_HOME/` called `endorsed`.
○ Place a copy of `$XALAN_HOME/serializer.jar` in `$CATALINA_HOME/endorsed/`.
○ Place a copy of `$XALAN_HOME/xalan.jar` in `$CATALINA_HOME/endorsed/`.
○ Place a copy of `$XALAN_HOME/xercesImpl.jar` in `$CATALINA_HOME/endorsed/`.
○ Place a copy of `$XALAN_HOME/xml-apis.jar` in `$CATALINA_HOME/endorsed/`.
○ Define an alias called `tomcat` whose value is `"$CATALINA_HOME/bin/catalina.sh run"`.[27]

⊗ Stop crossing off circles.

---

[25] On Windows, you can create a batch file (in any directory in your `PATH`) called `ENVIRONMENTCHECK.BAT` containing the following two lines.

```
@echo off
java org.apache.xalan.xslt.EnvironmentCheck
```

[26] On Windows, you can create a batch file (in any directory in your `PATH`) called `XALAN.BAT` containing the following seven lines.

```
@echo off
:rep
shift
set args=%args% %0
if not !%1==! goto rep
java org.apache.xalan.xslt.Process %args%
set args=
```

[27] On Windows, you can create a batch file (in any directory in your `PATH`) called `TOMCAT.BAT` containing the following two lines.

```
@echo off
"%CATALINA_HOME%"\BIN\CATALINA.BAT run
```