6.170 Laboratory in Software Engineering
Fall 2005
Final Project: RSS Client
Due: See Schedule

Contents:

Note: It has been brought to our attention that some 6.170 students have acquired Repetitive Strain Injuries (RSI) over the course of the final project in the past.

Introduction

There are two final project choices this term. For your final project, you can choose to work on either **RSS Client** or **Gizmoball**.

This handout describes the **RSS Client**. For information on **Gizmoball**, the other project, please refer to the 6.170 projects section.

If you choose **RSS Client** as your final project, your project will be to design, document, build, test, and release a standalone, desktop RSS client in Java.

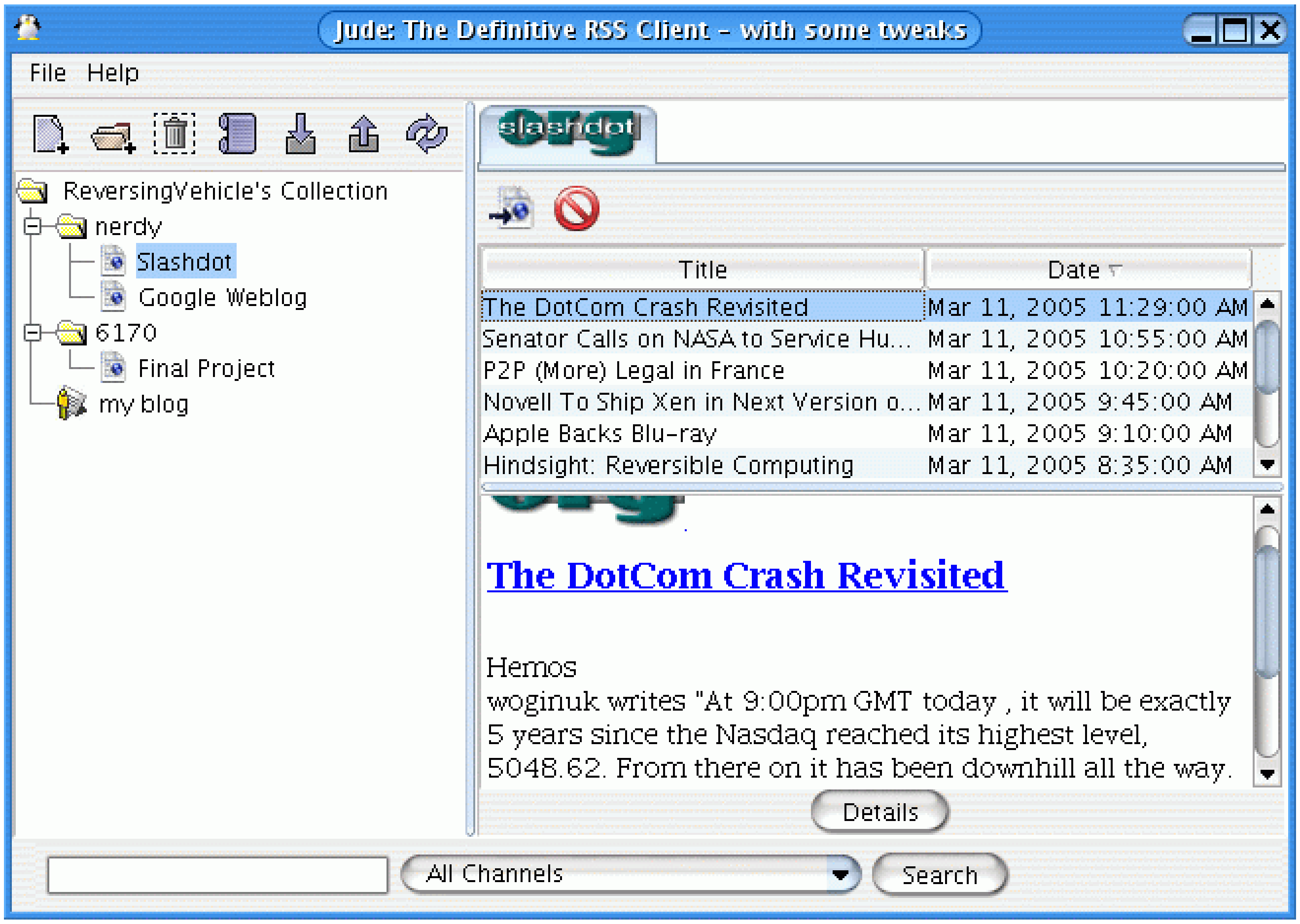
Required Features

In short, an RSS client downloads news *articles* and displays them. The articles are organized into *feeds*. A feed is identified with a URL, and, at any one time, accessing that URL returns a file containing the articles. Each article typically has a title, a summary, and a link to a web page for further information. The feed file is formatted in XML, using one of several RSS standard formats. (To see an example of a feed, look at http://www.slashdot.org/index.rsswhich is an RSS 1.0 feed for http://www.slashdot.org/.) You can also try Google's online RSS reader for free.

As new articles enter the feed, older articles are typically dropped out. Different sources update their feeds at different rates; a client usually polls the feed to determine whether it has been updated. RSS clients therefore have elements in common with browsers, newsgroup readers, and email and instant messaging clients.

This project is very open ended; you are free to develop your RSS client with your own choice of features. Nevertheless, a base set of features is required:

- **Display of articles**. The user can display the titles and summaries of articles in a feed, as well as the full articles (which will in general require rendering webpages). A visual indication shows which articles have been read, and users have the option of only displaying articles that have not yet been read. The client must be capable of reading feeds in the 3 most common formats (RSS 1.0, RSS 2.0 and Atom 0.3).
- **Subscription to feeds**. The user can enter feed addresses, associate informal names with them, and organize them into groups (usually called *channels*). The user can select the update period for individual feeds, which determines how often the client polls them. The collection of feeds, their names, organization into channels, and update periods can be saved as a file, or loaded from a file. In addition, the list of feeds alone (without organization into channels or user preferences) can be saved and loaded in a standard interchange format known as OPML (see below), so that users can share subscriptions.
- **Caching of articles**. When the machine on which the client is running is offline, the user can access articles that have been previously downloaded. It is not necessary to make all articles that have been downloaded in the past accessible; you can invent some reasonable scheme for retiring old articles, ideally under user control.
- **Basic keyword search**. The user can search amongst previously downloaded articles by keywords appearing in titles.
- **Support for the amendment**. When the amendment is released (see below), it will amend the requirements listed here, so your final project must comply with the requirements of the amendment, as well.



Title	Date
The DotCom Crash Revisited	Mar 11, 2005 11:29:00 AM
Senator Calls on NASA to Service Hu...	Mar 11, 2005 10:55:00 AM
P2P (More) Legal in France	Mar 11, 2005 10:20:00 AM
Novell To Ship Xen in Next Version o...	Mar 11, 2005 9:45:00 AM
Apple Backs Blu-ray	Mar 11, 2005 9:10:00 AM
Hindsight: Reversible Computing	Mar 11, 2005 8:35:00 AM

The DotCom Crash Revisited

Hemos
woginuk writes "At 9:00pm GMT today , it will be exactly 5 years since the Nasdaq reached its highest level, 5048.62. From there on it has been downhill all the way.

(example client)

Awards

Each team may enter its RSS client into a class contest for one or more of the following awards:

- Best design: awarded to the solution with the best design at the source-code level.
- Most usable: awarded to the solution with the best user interface design.
- Best feature: awarded to the solution with the most imaginative new feature.

Grading, Deliverables and Schedule

You'll do your project in phases, with the following milestones:

Phase	Deliverables	Due date
Preliminary Design	Preliminary design document	Wed, Nov 9th at 12 noon
Preliminary Release	Source code, specifications, unit tests	Mon, Nov 21st at 12 noon
Final Release	Final design document, source code, specifications, unit tests, user manual, webstart	**Mon, Dec 12th at 9:00 AM**

	packaging	

Each team will receive a single grade for the final project, determined as follows:

Category	Deliverable	Due date	% project grade	Graded on
Design	Preliminary design document	Wed, Nov 9th at 12 noon	15%	Are key issues identified?
	Final design document	**Mon, Dec 12th at 9:00 AM**	15%	Is design clean, robust and flexible?
Team work	Weekly meetings	weekly	10%	Did team work well together? Did all members participate constructively?
Packaging	User Manual	**Mon, Dec 12th at 9:00 AM**	8%	Is the tool easy to use? Is the user manual clear and helpful?
	Webstart delivery of executable	**Mon, Dec 12th at 9:00 AM**	2%	Is the client packaged correctly?
Implementation	Specifications, preliminary	Mon, Nov 21st at 12 noon	5%	Are important interfaces identified and crucial parts well documented in Javadoc?
	Specifications, final	**Mon, Dec 12th at 9:00 AM**	5%	Are important interfaces well documented in Javadoc?
	Unit tests, preliminary	Mon, Nov 21st at 12 noon	5%	Are there good unit tests for non-trivial classes?
	Unit tests, final	**Mon, Dec 12th at 9:00 AM**	5%	Are there good unit tests for non-trivial classes?
	Source code, preliminary	Mon, Nov 21st at 12 noon	15%	Is the code clean and well structured? Basic functionality working?
	Source code, final	**Mon, Dec 12th at 9:00 AM**	15%	Is the code clean, well-structured and low in defects?

Here is what each of the deliverables should contain:

- **Preliminary design document**: includes basic models (object model of problem domain, object model of code, module dependency diagram), rationale (succinct informal narrative explaining why certain design decisions were made, and what alternatives were considered and rejected); and work allocation and milestones (how tasks will be divided amongst team members, and the dates on which tasks are expected to be completed). It should be possible to read the document linearly, so you should minimize the use of forward references, and should provide enough overview and explanatory material to make the design artifacts comprehensible. You can assume that your reader is familiar with the materials you have been provided with, and understands the modelling notations. The purpose in preparing the preliminary design is to identify and explore important issues, so the document will be judged on how well it does this, rather than on the quality of the design per se.

- **Final design document**: includes basic models (object model of problem domain, object model of code, module dependency diagram), rationale (succinct informal narrative explaining why certain design decisions were made, and what alternatives were considered and rejected); a brief description of your strategy for testing and validation; and a post mortem (a discussion of which design decisions turned out well, and which turned out badly, and why; whether the milestones were met, and what you did when they were not). The same considerations about explanatory content apply here. Any changes between the preliminary and final design should be noted and explained.
- **User manual**: a standard user manual, intended for users not familiar with any of the course material. The better the design of the user interface of your client, the less you will need to explain in the user manual.
- **Specifications**: every public method should have at least a minimal Javadoc specification. A careful pre-post specification should be written for any subtle or important method.
- **Unit tests**: each test should have a brief comment explaining its purpose.
- **Source code**: will be judged by its correctness, clarity of structure, and the judicious use of runtime assertions and representation invariants.

Some general points:

- The work you hand in for the preliminary design and release should differ from that of the final release in its state of completeness, not in its quality. It's very important to get into the habit of working methodically. If you just hack like mad, and hope to make your code clean and elegant at the end, you won't succeed.
- For the preliminary release, you will be expected to demonstrate some basic functionality. It **must have** at least the following features:
  - Support at least one of the three required feed formats.
  - Be able to subscribe to a new feed that uses that format.
  - Be able to read and display articles from that feed.
  - Be able to close and re-open the application and read previously acquired articles without downloading.
- Your TA will judge the usability and correctness of your client largely during a demo at the end of term. You will have about 15 minutes to show off your work, to be followed by about 30 minutes of questions and discussion directed by your TA.
- After the preliminary release, we will give you an **amendment**: a request for an additional feature. How easy it is to accommodate it will depend on well you have designed your client to anticipate reasonable increments of functionality. In judging your final release, the new feature will be considered one of the required features.

- All deliverables should be handed in electronically and (with the exception of the code) as hardcopy to your TA (double sided and stapled). Late handins will be heavily penalized; for the final release, because of end-of-term constraints, late handins will not be accepted.

Weekly Meetings with TA

Each team will meet with its TA once a week for an hour. The official time for these meetings is during the regular 2-3pm class time, or the time you have been using for grading meetings. To receive full participation credit:

- All team members must be present at all meetings.
- All team members must answer questions and participate in the discussion at each meeting.
- A clear progress document that is useful for productive discussions must be handed in each week at the meeting. At the first meeting a draft of the Preliminary Design will serve as the progress document.

Although the progress document must be clear, it is short and informal. This document will form the basis of discussion during the meeting, and the TA will keep it on file as a record of progress made. The team should bring multiple copies to the meeting, one for each team member and one for the TA. This progress document should include the following information:

- A description of all the new issues that have been discovered during the previous week. This includes both a list of newly discovered bugs, and a list of unresolved design issues.
- A description of all the issues that have been solved over the past week. This includes a list of bugs that were fixed, and how they were fixed, and a list of design issues that were resolved, and how they were resolved.
- A list of all the issues from previous weeks that are still unresolved.
- A plan for the next week, with specific actions and goals for each team member.
- An assessment of success at meeting the previous week's plan.
- The document may also contain any other material that you feel describes your progress, such as object model or MDD fragments showing changes to the design.

Resources

This section is full of information and links that will help you complete your final project.

Third-Party Jars and Tools

Here we provide you with a list of third-party jars that you may want to use in your final project. If there are other jars that you are considering using, then please email the staff to get our permission first. The list is arranged so that the jars that are most likely to help you appear first, and those that help with features that are not required appear later. Thus, jars that help with parsing XML and RSS appear at the top of the list, whereas jars that help with sending instant messages and querying Google appear at the bottom of the list. You do not have to worry about the jars at the bottom of the list unless you plan on implementing advanced features in your RSS client.

Be forewarned that all of the jars below are free and open-source. Thus, the quality of the documentation that accompanies these jars will vary greatly. It may be sparse, or even incorrect. When choosing to use a third-party jar, be sure to do your homework and run some tests to make sure that the jar implements the functionality that it claims to provide before committing to using it. We recommend that you take advantage of the 6.170 online forum to discuss issues that arise in using these jars. Hopefully this will prevent your classmates from falling into the same traps that you did. As was the case for the first half of the semester, the 6.170 staff will look favorably upon students/groups who contribute to the forum.

Also, we realize that this list of tools is very long, and that you may only find a few of them helpful in completing your project. This is fine -- there is a cost that comes along with learning a new tool, and your time is limited, so pick your tools wisely. This list strives to give you an idea of what tools are out there, so that even if you do not use a tool on this project, then at least you will be aware of it when you have a use for it on another project.

Parsing XML

Because RSS, Atom, and OPML are all XML formats, your project will need a module that can parse XML. We do not expect, or recommend, you to write your own XML parser, so please take advantage of Xerces or your favorite Java library for parsing XML.

Xerces Java Parser (for XML)
http://xml.apache.org/xerces-j/
This library provides both a DOM and a SAX parser for XML. If you do not know what the difference between these parsers is, you may want to look at this comparison to learn more about them. Basically, if your XML file is not large or you are using most of the information in the document, then you probably want to use the DOM parser. As most of the XML files that you will be dealing with fall into one of these categories, you can probably get away with exclusively using the DOM parser.

XPath Tutorial
http://www.w3schools.com/xpath/
Using XPath is equivalent to writing regular expressions to extract pieces of information from an XML document. The benefit is that XPath expressions are often much easier to read and to write than regular expressions. Read this XPath tutorial to see if XPath is for you. If so, then you can use Xerces's support for XPath.

Xalan-Java XSLT Processor
http://xml.apache.org/xalan-j/
XSLT is a language for translating XML documents. For example, if you wanted to translate a document written in RSS 1.0 to a document in RSS 2.0, then XSLT would be a good tool to use. You may not find a use for XSLT in this project, but it is important to be aware of its existence if you find yourself writing applications that rely heavily on XML.

Parsing RSS

Parsing RSS feeds in various formats is one of the fundamental tasks that your client must be able to do. Thus, you may want to try using a third-party library to do this for you, as this could save your group a substantial amount of development time. None of these libraries is well-established, so make sure that the time saved by using one of these libraries will be greater than the time spent integrating it with your project.

Rome
https://rome.dev.java.net/
Rome is a set of Atom/RSS Java utilities developed by Sun. In our estimation, the Javadocs are not so good; however, the sample code they provide should be sufficient to allow you to take advantage of this library, if you choose to use it.

Informa
http://informa.sourceforge.net/
Informa is an open source project that aims to provide a suite of Atom/RSS Java utilities. Though the Informa home page claims that their library provides a lot of functionality, we do not believe that their implementation lives up to its promises, so be wary of that if you try to use it.

RSSLib4J
http://rsslib4j.sourceforge.net/
RSSLib4J is yet another open source project that aims to provide support for extracting information from an RSS feed. It does not claim to provide support for the Atom format, and as it is smaller in scope, it may be more complete.

Rendering HTML

Many RSS feeds provide excerpts from articles with URLs to the full article. (If they provided the full article in the feed, then no one would look at the advertisements on their web sites. Also, it would put a larger load on their servers, which already suffer enough from news aggregators [Ano04].) Thus, most RSS clients provide the ability to navigate from the excerpt to the web site, so the RSS client needs to be able to render an HTML web page. Again, we do not expect, or recommend, that you write your own web browser, so consider using one of the following widgets for displaying web pages:

JDesktop Integration Components (JDIC)
https://jdic.dev.java.net/
JDIC is a suite of Java components aimed at "provid[ing] Java applications with access to facilities provided by the native desktop such as the mailer, the browser, and registered document viewing applications." You will probably be interested in the `WebBrowser` component that JDIC provides, as it will allow you to embed a mature web browser inside a Swing application.

JRex
http://jrex.mozdev.org/
JRex is similar to JDIC in that it wraps the Mozilla web browser, making it possible to embed Mozilla in a Swing application. The staff has found JDIC easier to use than JRex, but you may want to experiment and decide for yourself.

javax.swing.JEditorPane
http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JEditorPane.html
This is the widget that comes with Sun's SDK for rendering HTML in a Swing application. It is simple to use and will probably make it easier to deploy your application via Java Web Start; however, its ability to render web pages is rather poor by the standards of modern web browsers.

SWT Browser Widget
http://www.eclipse.org/swt/
SWT is a windowing toolkit that makes it possible to embed native applications inside your GUI. Though it is possible to create a GUI that uses both SWT and Swing components, it is somewhat of a headache, so if you commit to using this widget, then you will probably want to write your entire application in SWT.

Project Management

The final project is going to require group coordination, so you may find the following tools helpful in managing your project.

Ant
http://ant.apache.org/
You have already used Ant this semester in some of the problem sets. Now that you are working on a more involved project, you may want to write Ant scripts to automate other common tasks, such as signing jarfiles or uploading a local version of a web page to a server.

Maven
http://maven.apache.org/
Maven is a project management tool that creates a web site with reports and other information about your project. For an example of a web site that Maven would produce, just look at the Maven home page, as the Maven developers have learned to "eat their own dog food".

Log4j
http://logging.apache.org/log4j/
Many students get into the bad habit of littering the code with `System.out.println` statements when trying to debug their code. Over time, these statements get commented in, commented out, deleted, rewritten, etc. This adds up to a lot of wasted time. Using a tool for logging messages, such as Log4j, can help stop the `println` madness.

Bug Tracking

Throughout the development of your project, your application will likely have some bugs. Once a bug is discovered, it is important to make a record of it in some way and then update the record when it is fixed. You do not have to use one of these two tools. But you must formally record every bug, document what problem is caused by the bug, then record how you fixed it (if the bug is indeed fixed).

Bugzilla
http://www.bugzilla.org/
Bugzilla is probably the most well-known bug tracking system, as it is used to track the bugs in the Mozilla web browser (as well as in the other Mozilla products). This is often considered a behemoth of a bug tracker, but it is free for you to use, if you like.

Flyspray
http://flyspray.rocks.cc/

Flyspray is a simpler bug tracker than Bugzilla, and some 6.170 students have used it in the past for their final project with great success.

Databases

If you want to build an RSS client that is able to manage a large number of articles, then you may want to store the articles in a local database on the user's machine rather than in a directory full of text files. Below is a short list of Java databases that are small enough that it would be reasonable to deploy one of them with your RSS client. If you do not have prior experience with SQL, then learning about database design while trying to implement your final project may not be a judicious use of your time. Again, you do not have to integrate your RSS client with one of these databases unless you want to.

Derby/Cloudscape
http://db.apache.org/derby/
This is a lightweight, but powerful database that was developed by IBM (under the name Cloudscape) and was recently donated to the Apache Software Foundation (under the name Derby).

Hibernate
http://www.hibernate.org/
Hibernate is another database that plays well with Java, both on the desktop and on web servers.

AOL Instant Messenger (AIM)

If you wanted your RSS client to send you an instant message via AIM when it discovered an article on a particular topic, then you may want to use one of these libraries.

Daim
https://daim.dev.java.net/
It lets you programmatically use AIM through the OSCAR protocol.

JAIMBot
http://jaimbot.sourceforge.net/
It lets you programmatically use AIM through the TOC protocol.

Miscellaneous

Below are some other interesting libraries. We're not sure exactly what you would use them for in an RSS client, but we suspect that a group of 6.170 students could do something clever with them if they tried.

Google Web APIs
http://www.google.com/apis/
The Google Web APIs let you programmatically query Google up to 1000 times per day, provided you have a Google Account and a license key (both of which are free).

Lucene
http://lucene.apache.org/java/docs/
Lucene is a text search engine. If you want to let users search the bodies of all the

articles in your RSS client, then Lucene is probably the way to go. Lucene can manage its own storage, or it can work with a third party database, such as the databases listed above under the Databases section.

FreeTTS
http://freetts.sourceforge.net/
FreeTTS is a Java library that converts text to speech. Thus, you could use FreeTTS to create audio files of a robotic voice reading your RSS feeds.

RSS and BitTorrent
http://blogs.law.harvard.edu/tech/bitTorrent
This article explains how RSS 2.0 has support for enclosures. An **enclosure** is an element in the an RSS file that points to a file related to the article that the user may want to download, such as a movie or image file. Thus, if you subscribe to a feed that encloses `.torrent` files, then you could add a feature to your RSS client to "download every `.torrent` enclosed in an article about X"!

Java Web Start (JAWS)

When releasing a software product, it is important to package it so that it is simple for users to install and to run. To this end, Sun Microsystems has created Java Web Start (JAWS) to simplify the deployment of Java programs. Thus, you are required to to package your final project using JAWS so that you can easily share your final project with the world (or at least with the 6.170 staff).

The Problem That JAWS Solves

One of the problems with Java is that it is difficult for the average person to run a Java desktop application because that requires downloading and installing the Java Runtime Environment (JRE). Even after the JRE is installed, executing a Java program may require setting the classpath and including native libraries. Obviously, these are not tasks that an average user will be able to do.

One solution [for Windows] is to fork over $200 to buy a static compiler like Excelsior Jet that will convert your .jar into a Windows executable (.exe) file that people can just double-click on and run like any other Windows application. Obviously, this is a little expensive and it only solves the problem for one platform.

A cheaper, platform-independent solution is JAWS. Basically, anyone who has JAWS installed can install a JAWS application by clicking on a hyperlink in a web browser to "download" it. When clicked, JAWS downloads the application to the client's machine so that he need not be online to run it in the future. However, if the client is online when he subsequently launches the application, JAWS will ping the web site from which it was downloaded to see if there have been any updates to the application since it was downloaded. If there are, it silently gets them from the web site, so you can be sure that your client is always running the latest version of your software. Thus, all a user needs to do is click on a hyperlink in a web browser then hit **OK** in a few dialog boxes to get your application up and running.

Configuring Your Application for JAWS

The JAWS developer site is pretty thorough about how to get your application working with JAWS. The basic idea is that you upload your jarfile, and any jarfiles on which it depends, to a web site. In that directory, you also create an XML descriptor of your application that identifies which jars it uses. This file is called a **JNLP** file. People can then download your application by clicking on a hyperlink to your JNLP file.

When the user clicks on that hyperlink, the web browser would know to associate the .jnlp file extension with JAWS, so it will start JAWS and have it parse the JNLP file. Then JAWS will look at the file to figure out which jarfiles to download and then launch your application locally on the client's machine.

Thus, you do not need to modify your Java code to make your application work with JAWS. Your only responsiblity is to create the JNLP file and put it on a website.

Gotchas: If your application requires special permissions, such as writing files on the client's machine or using native libraries, then you may have to address some of the issues in the JAWS Gotchas section below.

See JAWS in Action

JDIC is one of the third-party libraries that we recommend above for rendering HTML in your RSS client. If you look under the Demos section on the JDIC web site, you will see a hyperlink labeled **Browser** that you can click on to run the demo. Clicking on this link should launch JAWS, and then the demo. Launching your RSS client from a web page should be just as easy for your users.

JAWS Gotchas

One of the tricky parts about using JAWS to deploy Java software is that you probably need to sign your jars. When you sign a jar, JAWS asks the user if he or she trusts your digital signature before running the code. If the user trusts you, then your application can do things like read and write files on the user's disk. If you do not sign your code, then you will get a SecurityException when your application tries to do such naughty things. Because your RSS client will likely persist data by writing files to the user's machine, you must sign your code if you want your users to have the full functionality of your application. Also, every time you rebuild your jar, you have to re-sign your code, so you should write a script or an Ant task to automate the code signing process; otherwise, it's pretty tedious to do it yourself every time you make a change to your jar.

Finally, let it be known that JAWS is not a perfect solution in that not every user has it. Some browsers/operating systems come with it pre-installed, in which case it should be simple for your users to get your JAWSified application. However, other users may not be as fortunate, so you may want to provide a link where users can download JAWS before getting your application. Generally, users hate downloading things, but that's what's so great about JAWS -- it's so sneaky! A user just clicks on a hyperlink as if it's going to take him to some web page, and then suddenly it downloads your RSS client and then runs it! It's beautiful. (And it's much easier than trying to explain to your users how they need to get the JRE and then add java.exe to their PATH...)

Outline Processor Markup Language (OPML)

OPML is an XML format for outlines. Though OPML may be used to describe outlines in general, it is most often used for sharing lists of RSS feeds. Your RSS client will be required to import and export a list of feeds in this format. Here is an example of an OPML 1.0 file with a list of this semester's 6.170 forum feeds:

```
<?xml version="1.0"?>
<opml version="1.0">
    <head>
        <title>6.170</title>
    </head>
    <body>
        <outline type="rss" text="Problem Set 1"
xmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/feeds1.xml"/>
        <outline type="rss" text="Problem Set 2"
xmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/feeds2.xml"/>
        <outline type="rss" text="Problem Set 3"
xmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/feeds3.xml"/>
        <outline type="rss" text="Problem Set 4"
xmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/feeds4.xml"/>
    </body>
</opml>
```

The OPML example above does not include every possible tag that is permissable by the OPML 1.0 Specification. For example, some RSS clients may include additional attributes in the tags and additional custom-defined tags, or they may even claim to be OPML 1.1:

```
<?xml version="1.0"?>
<opml version="1.1"
xmlns:fd="http://www.bradsoft.com/feeddemon/xmlns/1.0/">
    <head>
        <title>ourFavoriteFeedsData.top100</title>
        <dateCreated>Fri, 02 Jan 2005 12:59:58 GMT</dateCreated>
        <dateModified>Fri, 23 Jul 2005 23:41:32 GMT</dateModified>
        <ownerName>Dave Winer</ownerName>
        <ownerEmail>dave@userland.com</ownerEmail>
        <expansionState></expansionState>
        <vertScrollState>1</vertScrollState>
        <windowTop>20</windowTop>
        <windowLeft>0</windowLeft>
        <windowBottom>120</windowBottom>
        <windowRight>147</windowRight>
    </head>
    <body>
        <outline type="rss" version="RSS"
            text="Problem Set 1" title="Problem Set 1"
description="Problem Set 1"
            xmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/feeds1.xml"
            htmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/forum1/"
        />
        <outline type="rss" version="RSS"
```

```
            text="Problem Set 2" title="Problem Set 2"
description="Problem Set 2"
            xmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/feeds2.xml"
            htmlUrl="http://courses.csail.mit.edu/6.170/old-www/2005-
Fall/forum2/"
        />
    </body>
</opml>
```

When importing OPML: Your client is not required to understand these additional tags and attributes when importing an OPML file. But it should not reject a file that includes these additional data. In other words, your client should be **forgiving** when importing an OPML file. If an OPML file is well-formed, your RSS program must be able to parse it and read in its contents even though your program may not understand all the information available in the file. Furthermore, when you see OPML 1.1 file, you should read it as if you are reading an OPML 1.0 file. (Since the OPML specification is so vague, there is no practical difference between the two versions).

When exporting OPML: Any OPML file created by your RSS program must be well-formed according to the OPML specification. Your program does not have to use every possible tag that the specification allows. It only has to provide enough information so that when a user tries to import an OPML file generated by your RSS program into their RSS program, the user's program will be able to find an **xmlURL** and **text** attribute for each feed.

For those of you familiar with HTML, you may notice that OPML bears a strange similiarity to HTML, even though the two file types have nothing to do with one another. Though there is a number of peculiarities with the OPML format, it has been popularized by Radio Userland, and a number of RSS tools have been built around its specification. Thus, your RSS client is required to support OPML to increase its interoperability with other products.

Specifications for Various RSS Formats

This section contains links to the specifications for the three RSS formats that your client is required to support. Your client need only recognize these three format, but ideally, it should be able to parse any file that is validated by http://www.feedvalidator.org/.

RSS 2.0 Specification
http://blogs.law.harvard.edu/tech/rss
The RSS 2.0 specification was written by Dave Winer. Note that the web site for the RSS 2.0 specification claims that RSS stands for **R**eally **S**imple **S**yndication.

RSS 1.0 Specification
http://web.resource.org/rss/1.0/spec
The RSS 1.0 specification was written by various people from various organizations. Note that the web site for the RSS 1.0 specification claims that RSS stands for **R**DF **S**ite **S**ummary.

Atom 0.3 Specification
http://www.atomenabled.org/developers/syndication/atom-format-spec.php
Although the site notes that the current draft "HAS NOT been submitted for

publication, and does not have any status," it provides enough detail for you to be able to create a parser that can read the Atom format.

References

The articles cited in this document provide additional background information on topics related to this project.

[Ano04] Anonymous. "When RSS Traffic Looks Like a DDoS.", July 20, 2005.

[Bol04] Michael Bolin, "What is Site Syndication?"

[Oba03] Dare Obasanjo, "Building a Desktop News Aggregator.", March 14, 2003.

[OPML] Wikepedia. What is OPML?.

[Pil02] Mark Pilgrim, What is RSS?, Dec 18, 2002.

[Wal98] Norman Walsh, A Technical Introduction to XML, Oct 3, 1998.

Clarifications and Errata

No clarifications or errata are provided.