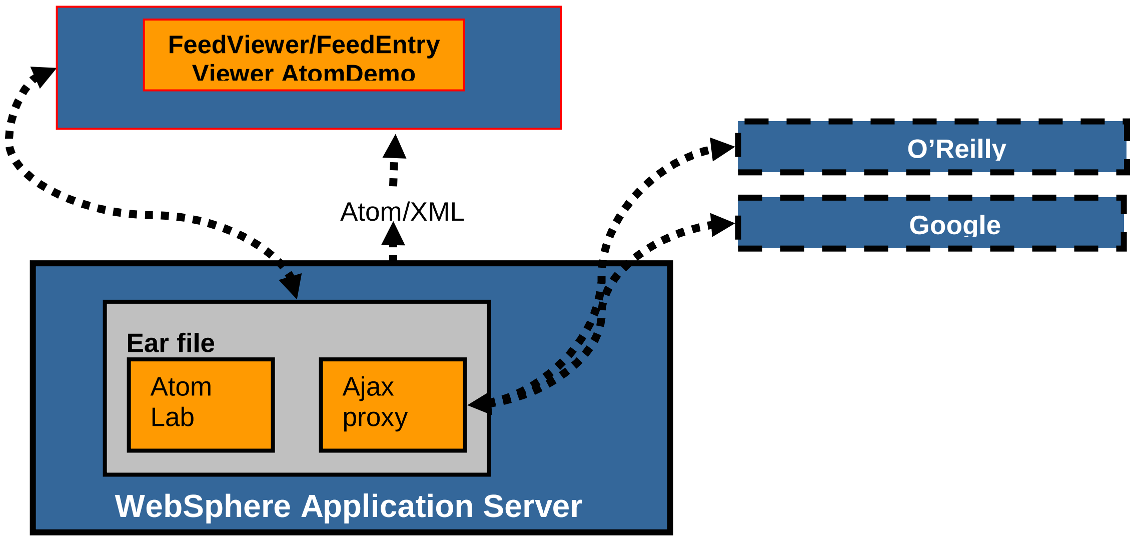
IBM WebSphere® Application Server Feature Pack for Web 2.0 – LAB EXERCISE

Atom lab

What this exercise is about

In this exercise you are going create an application that deals with feeds. The application shows off the Atom library, Atom widgets and the Ajax proxy that are included in the IBM WebSphere Application Server Feature Pack for Web 2.0. You will use Atom feeds and view them through a nice format using an Ajax application consists of Dojo Toolkit widgets and the Atom Library and FeedViewer widgets.


FeedViewer/FeedEntry Viewer AtomDemo
O'Reilly
Google
Atom/XML
Ear file
Atom Lab
Ajax proxy
WebSphere Application Server

Lab requirements

For this lab your system must have Rational Application Developer 7.5 installed and an associated WebSphere Application Server 6.1 test environment.

What you should be able to do

At the end of this lab you should be able to:

- Create an application in Rational Application Developer
- Have a better understanding of why the Ajax proxy is needed and how to use it.
- Have a better understanding of how to incorporate the Dojo Toolkit and the IBM Extensions to the Dojo Toolkit – Atom Library into an application.

Introduction

The Atom library: This library contains three different features. First are the general utility functions to support the rest of the library. Next is the data model for the various parts of Atom, such as content, person, link, feed, and entry. These data models are then used to define the AtomIO object, which is a wrapper to the various functions intrinsic to Atom feeds and the Atom Publishing Protocol (APP).

The Atom Widgets: Included in this Feature Pack for Web 2.0 are three widgets. FeedViewer, displays the title and dates of entries in a feed. FeedEntryViewer, displays the details of a selected Atom Entry in a FeedViewer. FeedEntryEditor, similar to FeedEntryViewer, but allows for editing of existing entries and creation of new ones.

Atom limitations: Due to the same-origin policy present in the browser, JavaScript code cannot request resources from servers other than the server which the current script originated from. There are several workarounds, however the Atom library uses the standard XMLHTTPRequest. Because of this, the library can only handle feeds that originate from the same domain that the library is served from. To retrieve feeds from other domains, use the Ajax Proxy (included as part of this feature pack) to broker requests to external resources. The sandboxing policy that necessitates the need for a proxy is commonly referred to as the same-origin policy on most major browsers. As an example, if the JavaScript application originated from domain A and attempts to use an XMLHTTPRequest to domain B, then the browser prevents the domain B request. This browser design is intentional, as it is intended to protect users from malicious attacks that download script code from external servers or try to push user data to unauthorized external servers.

The Ajax proxy provided with the IBM WebSphere Application Server Feature Pack for Web 2.0 is a reverse proxy. You can install the reverse proxy into one or more application servers as a Web application. . Requests to the proxy are forwarded to the intended server. From the client's perspective, all of the proxied requests appear to originate from the same server, even though the reverse proxy might be forwarding requests to multiple Web servers. In other words, the proxy can be used to broker client requests from multiple domains while using Ajax. This allows Web application developers to judiciously bypass the JavaScript sandboxing rules that prevent the start of network requests to servers from where the JavaScript did not originate.

In this lab you will create a Java EE Ajax based application. The application will use the FeedViewer and FeedEntryViewer Atom widgets and will display the Atom feeds for two different URLs. The application will mashup content with another server, in particular, it will mashup the Atom feeds content from the two different URLs and display this information. The service information originates from the two other domains and provides the Atom feed data. The application will use the Ajax proxy to easily make available the two other services in different domains.

Exercise instructions

Instructions and subsequent documentation use symbolic references to directories which are listed as follows:

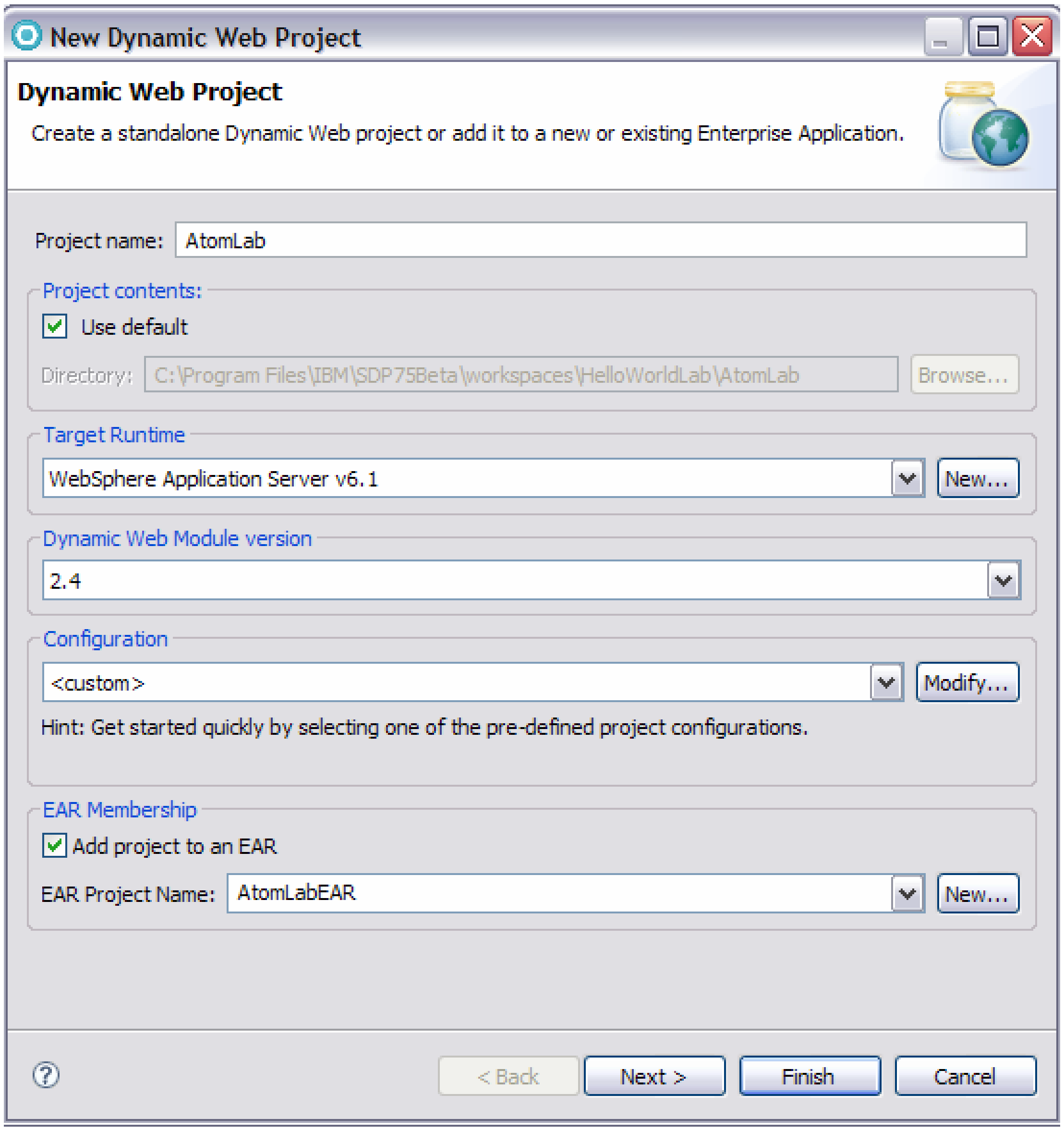
Reference Variable	**Windows Location**
<WAS_HOME>	C:\Program Files\IBM\SDP75\runtimes\base_V61

Part 1: Create a new application

____ 1. Start the Rational Application Developer. Go to **Start > All Programs > IBM Software Delivery Platform > IBM Rational Application Developer 7.5 > IBM Rational Application Developer**

____ 2. Create a new application, go to **File > New > Dynamic Web Project**

__ a. Enter **AtomLab** for the Project name. Choose the Target Runtime as WebSphere **Application Server V6.1**. For EAR Membership **checkmark** the Add Project to and EAR check box and enter EAR Project Name of **AtomLabEAR** as in the picture below:



Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: AtomLab

Project contents:
✓ Use default
Directory: C:\Program Files\IBM\SDP75Beta\workspaces\HelloWorldLab\AtomLab

Target Runtime
WebSphere Application Server v6.1

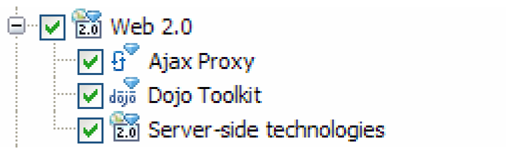
Dynamic Web Module version
2.4

Configuration
<custom>
Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR Membership
✓ Add project to an EAR
EAR Project Name: AtomLabEAR

__ b. Select the **Modify…** button in the Configuration section

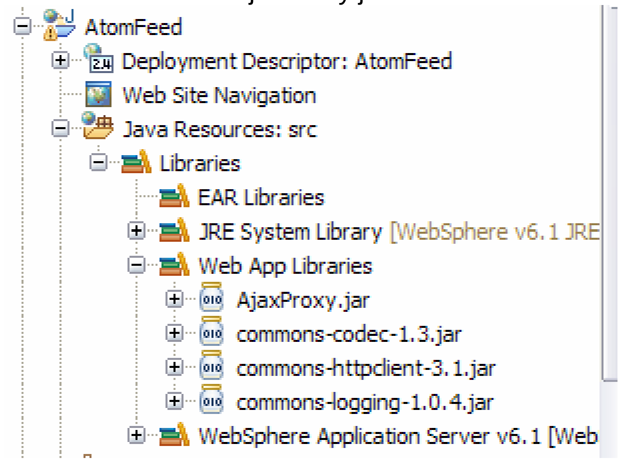
__ c. Expand the **Web 2.0** section and select **Ajax Proxy**, **Dojo Toolkit**, and **Server-side technologies**



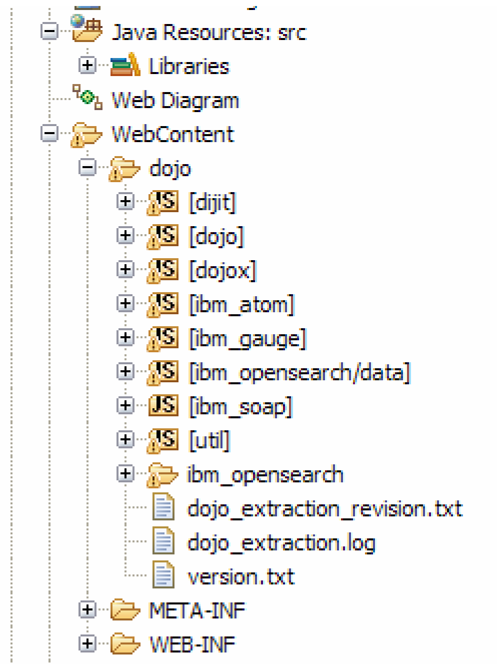
__ d. Select **Next > Next > Finish** to create your application

__ e. Your application will be created and Rational Application Developer will go and validate your new application. Note: you will see a lot of Warnings. These are warnings that are within Dojo, you can ignore these warnings.

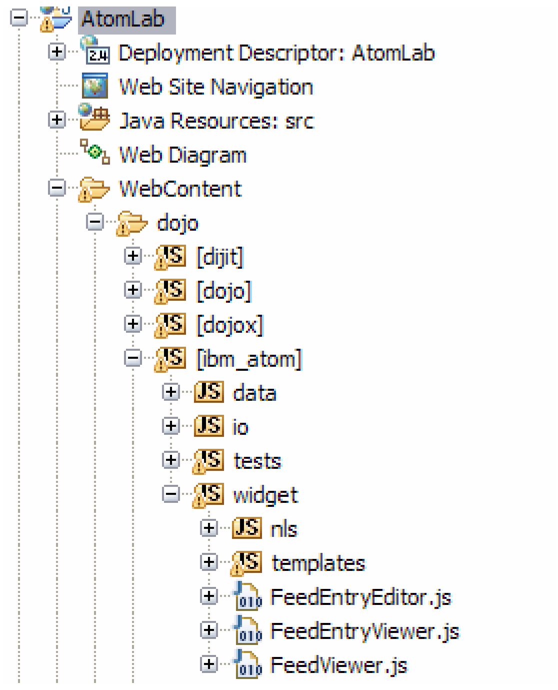
____ 3. Walk through your new application. First expand out **Java Resources: src > Libraries > Web App Libraries** and notice the AjaxProxy.jar file that is included from the Feature Pack for Web 2.0.


AtomFeed
Deployment Descriptor: AtomFeed
Web Site Navigation
Java Resources: src
Libraries
EAR Libraries
JRE System Library [WebSphere v6.1 JRE
Web App Libraries
AjaxProxy.jar
commons-codec-1.3.jar
commons-httpclient-3.1.jar
commons-logging-1.0.4.jar
WebSphere Application Server v6.1 [Web

____ 4. Expand the Web Content folder and then the dojo folder. Notice all the dojo imports and the IBM extensions to the Dojo Toolkit (ibm_atom, ibm_gauge, ibm_opensearch, ibm_soap).



____ 5. Next notice that the Atom widgets (FeedEntryEditor, FeedEntryViewer, and FeedViewer) included in ibm_atom. The Atom widget JavaScript files are located under **AtomLab > WebContent > dojo > ibm_atom > widget**. These Atom widgets are IBM Extensions to the Dojo Toolkit included in the Feature Pack for Web 2.0.



```
AtomLab
  Deployment Descriptor: AtomLab
  Web Site Navigation
  Java Resources: src
  Web Diagram
  WebContent
    dojo
      [dijit]
      [dojo]
      [dojox]
      [ibm_atom]
        data
        io
        tests
        widget
          nls
          templates
          FeedEntryEditor.js
          FeedEntryViewer.js
          FeedViewer.js
```

You can click tests > widget and right-click any of the test html files and Open with Browser or HTML Editor to view what the Atom widgets look like and how they are used.

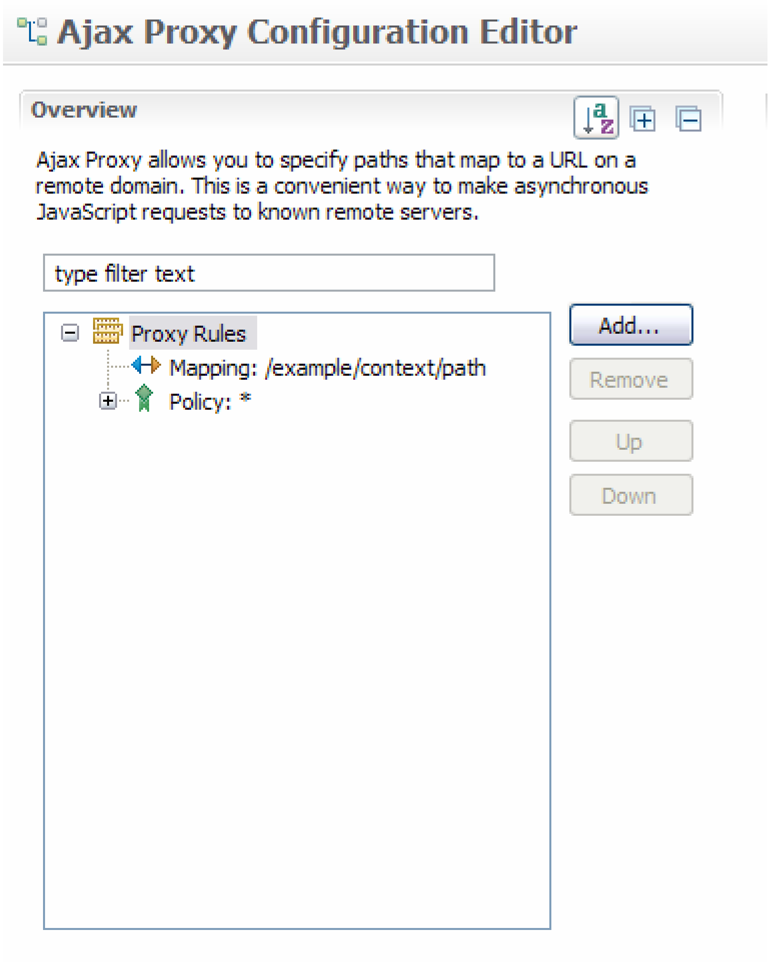
Part 2: Ajax proxy

As explained in the introduction to the lab, the Ajax proxy is needed due to the same-origin policy present in the browser. JavaScript code cannot request resources from servers other than the server which the current script originated from. Since the Atom library uses the standard XMLHTTPRequest, the library can only handle feeds that originate from the same domain that the library is served from. To retrieve feeds from other domains, you will introduce the Ajax Proxy (included as part of the Feature Pack for Web 2.0) to broker requests to external resources.

____ 1. Open proxy-config.xml located in AtomFeed > Web Content > WEB-INF

__ a. The proxy-config.xml file defines the policy by which URI requests are supported to pass through the proxy and how context paths from the client are mapped to the URI on a server.

__ b. Click the **Design** tab.



Ajax Proxy allows you to specify paths that map to a URL on a remote domain. This is a convenient way to make asynchronous JavaScript requests to known remote servers.

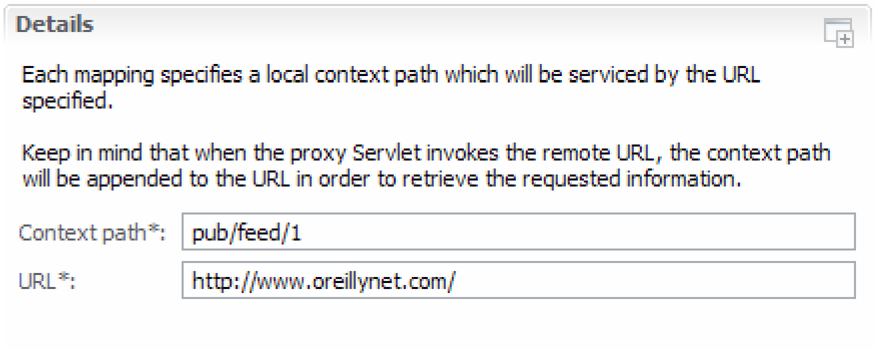
Proxy Rules
Mapping: /example/context/path
Policy: *

____ 2. Add the mapping for the O'ReillyNet site.

__ a. Click the **Add** button.

__ b. Select **Mapping** and enter

1) pub/feed/1 for the contextpath and http://www.oreillynet.com/ for the url



Each mapping specifies a local context path which will be serviced by the URL specified.

Keep in mind that when the proxy Servlet invokes the remote URL, the context path will be appended to the URL in order to retrieve the requested information.

Context path*: pub/feed/1

URL*: http://www.oreillynet.com/

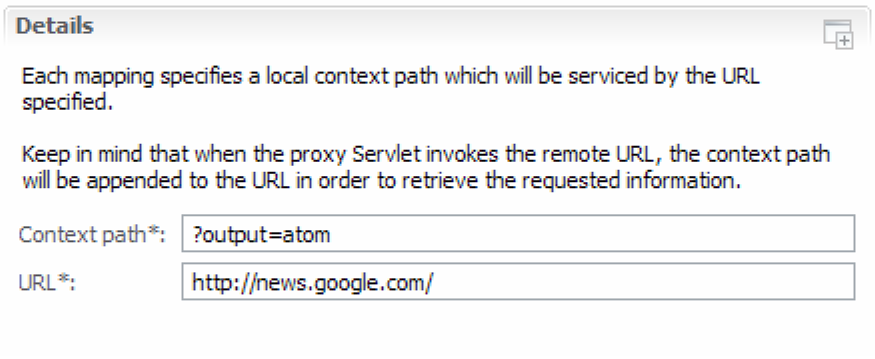
__ c. Select **Proxy Rules**

____ 3. Add the mapping for the Goggle site.

__ a. Click the **Add** button.

__ b. Select **Mapping**

1) ?output=atom for contextpath and http://news.google.com/ for url



Each mapping specifies a local context path which will be serviced by the URL specified.

Keep in mind that when the proxy Servlet invokes the remote URL, the context path will be appended to the URL in order to retrieve the requested information.

Context path*: ?output=atom

URL*: http://news.google.com/

__ c. Select **Proxy Rules**

____ 4. Click the example Mapping and select Remove

____ 5. Click **Source** to view the source of the proxy-config.xml you should see the two mappings that you added:

```
<proxy:mapping url='http://www.oreillynet.com/' contextpath='pub/feed/1' />
<proxy:mapping url="http://news.google.com/" contextpath="?output=atom" />
```

__ a. The contextpaths above are the two sites you will grab Atom feeds entries from.

____ 6. **Save** your proxy-config.xml file.

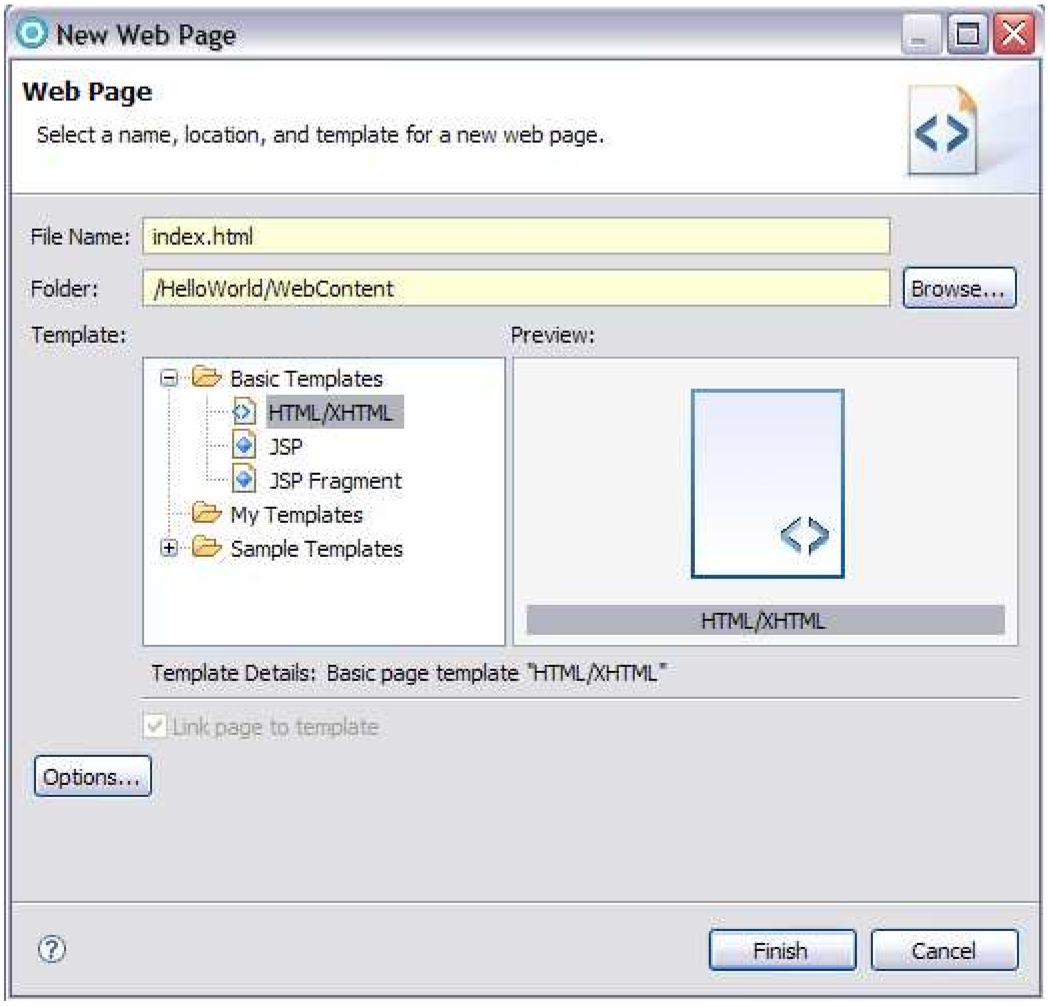
Part 3: Create client side code

The Feature Pack for Web 2.0 includes IBM Extensions to the Dojo Toolkit. One of these extensions is the ibm_atom. It includes the Atom Library, AppStore, and three widgets FeedViewer, FeedEntryViewer, and FeedEntryEditor. The FeedViewer widget is used to display multiple feeds. The FeedEntryViewer widget is used to display one selected feed. You will use these widgets in your application.

____ 1. Create a new Web page.

__ a. Right click the Web Content directory of your AtomLab Application and choose **New > Web page**.

__ b. Enter **index.html** for the File name and choose **HTML/XHTML** for the Basic Template, Click **Finish**.



Select a name, location, and template for a new web page.

File Name: index.html

Folder: /HelloWorld/WebContent

Template:

Preview: HTML/XHTML

Template Details: Basic page template "HTML/XHTML"

Link page to template

____ 2. You will now add the dojo code to create the client side of the application. You can use the drag and drop widgets that are a part of the Rational Application Developer or you can **replace** the index.html code with the following code instead of using the drag and drop palette.

This code is provided in AtomSnippet1.txt

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Demo of ATOM Feed Widgets.</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<style type="text/css">
       @import "dojo/dijit/themes/tundra/tundra.css";
       @import "dojo/ibm_atom/tests/widget/css/EntryHeader.css";
       @import "dojo/ibm_atom/tests/widget/css/HtmlFeedViewer.css";
       @import "dojo/ibm_atom/tests/widget/css/HtmlFeedViewerGrouping.css";
       @import "dojo/ibm_atom/tests/widget/css/HtmlFeedViewerEntry.css";
       @import "dojo/ibm_atom/tests/widget/css/HtmlFeedEntryViewer.css";
</style>
<script type="text/javascript">
  var djConfig = {parseOnLoad: true};
</script>
<script type="text/javascript" src="dojo/dojo/dojo.js"></script>
<script language="JavaScript" type="text/javascript">
    dojo.require("ibm_atom.io.atom");
    dojo.require("ibm_atom.widget.FeedViewer");
    dojo.require("ibm_atom.widget.FeedEntryViewer");
    dojo.require("dijit.layout.ContentPane");
    dojo.require("dijit.layout.TabContainer");
</script>
</head>
<body class="tundra">
    <div style="height: 400px; overflow: hidden; border-style: solid; border-width: 1px;">
       <div dojoType="dijit.layout.TabContainer" style="height: 100%; width: 100%;">
              <div dojoType="dijit.layout.ContentPane" title="O'Reilly Network Articles">
                       <div dojoType="ibm_atom.widget.FeedViewer"
                           url="proxy/pub/feed/1"
                           entrySelectionTopic="atomfeed/entry/topic">
                       </div>
                  </div>
              <div dojoType="dijit.layout.ContentPane" title="Google News">
                          <div dojoType="ibm_atom.widget.FeedViewer"
                              url="proxy/%3Foutput%3Datom"
                              entrySelectionTopic="atomfeed/entry/topic">
                          </div>
              </div>
        </div>
    </div>
    <br>
    <div style="height: 400px; overflow: auto; border-style: solid; border-width: 1px;">
           <div dojoType="ibm_atom.widget.FeedEntryViewer"
                id="feedEditor"
                enableMenu="true"
                enableMenuFade="true"
                enableEdit="true"
                displayEntrySections="title,authors,summary,content,id"
                entrySelectionTopic="atomfeed/entry/topic">
           </div>
    </div>
</body>
</html>
```

____ 3. Examine the code.

__ a. First notice the reference of the FeedViewer and FeedEntryViewer

1) dojo.require("ibm_atom.widget.FeedViewer");

2) dojo.require("ibm_atom.widget.FeedEntryViewer");

__ b. Next notice the **FeedViewer widget** calls.

1) For the O'Reilly Network Articles

```
<div dojoType="ibm_atom.widget.FeedViewer"
        url="proxy/pub/feed/1"
        entrySelectionTopic="atomfeed/entry/topic">
</div>
```

2) Notice for the URL in the code call above that you see the Ajax proxy context path for the http://www.oreillynet.com/ URL.

__ c. **The Feed Viewer Widget** is a reference widget for an Atom feed viewer implementation; this widget fetches a feed from the given URL and displays, in chronological order, entry titles and updated elements. This widget supports click events to highlight an entry and publishes the selected entry on the given topic.

Widget Variables

String	url	The URL from which to retrieve the Atom feed
String	entrySelectionTopic	The topic to both subscribe and publish. This enables two-way communication between the FeedViewer and FeedEntryViewer or FeedEntryEditor widgets.

__ d. View the other call to the FeedViewer widget

1) For the Google news feed

```
<div dojoType="ibm_atom.widget.FeedViewer"
            url="proxy/%3Foutput%3Datom"
            entrySelectionTopic="atomfeed/entry/topic">
</div>
```

2.) Notice the url does not appear quite the same as the context path definition. The reason it does not is that the Google news service context path is really a query parameter. This is because Google news requires a query parameter to output its feed in Atom format. In order to represent that query parameter as a context path when making an XHR call, it must be URL encoded. In the above URL, the encoding %3F is the ? character and the %3D is the = character. Since they are encoded, the Ajax Proxy can treat them as the context path and properly map it to the rule for the Google news service.

__ e. Notice how the **entrySelectionTopic** variable is the same for the two FeedViewer calls, entrySelectionTopic="atomfeed/entry/topic". This means that both feed viewer widget instances will publish feed selection events to the same topic. This means that multiple feed views can drive the display in a single FeedEntryViewer.

__ f. View the **FeedEntryViewer** code call

```
<div dojoType="ibm_atom.widget.FeedEntryViewer"
        id="feedEditor"
        enableMenu="true"
        enableMenuFade="true"
        enableEdit="true"
        displayEntrySections="title,authors,summary,content,id"
        entrySelectionTopic="atomfeed/entry/topic">
</div>
```

__ g. The **Feed Entry Viewer Widget** displays an individual entry element. A bar is provided at the top with a menu to turn specific elements on or off. The viewable elements can be dictated at creation time. In addition to the functions outlined in the table below, there are several additional functions, used mainly internally, to set the value of various nodes in the page.

Widget Variables

Boolean	enableMenu	Whether to enable the menu, allowing you to toggle the display of the entry elements.
Boolean	enableMenuFade	Indicates if a slick-looking fade effect is used for the menu described above.
Boolean	enableEdit	Indicates whether edit functions are enabled. This flag is only used if using the FeedEntryEditor widget, which extends this widget.
String	displayEntrySections	The entry elements to display as a comma-separated list. The available sections are title, authors, contributors, summary, content, ID, and updated.
String	entrySelectionTopic	The topic to both subscribe and publish. This enables two-way communication between the FeedViewer and FeedEntryViewer or FeedEntryEditor widgets.

__ h. Notice how the entrySelectionTopic is the same value as it was for the FeedViewer calls. This is because the FeedEntryViewer is listening to the topic that the two feed view widgets will publish to.

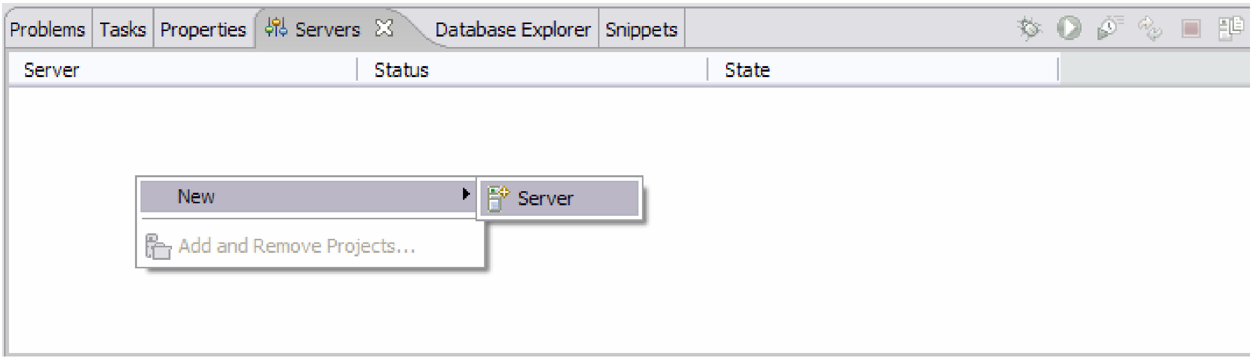
____ 4. **Save** the file.

Part 4: Define your WebSphere Application Server V6.1

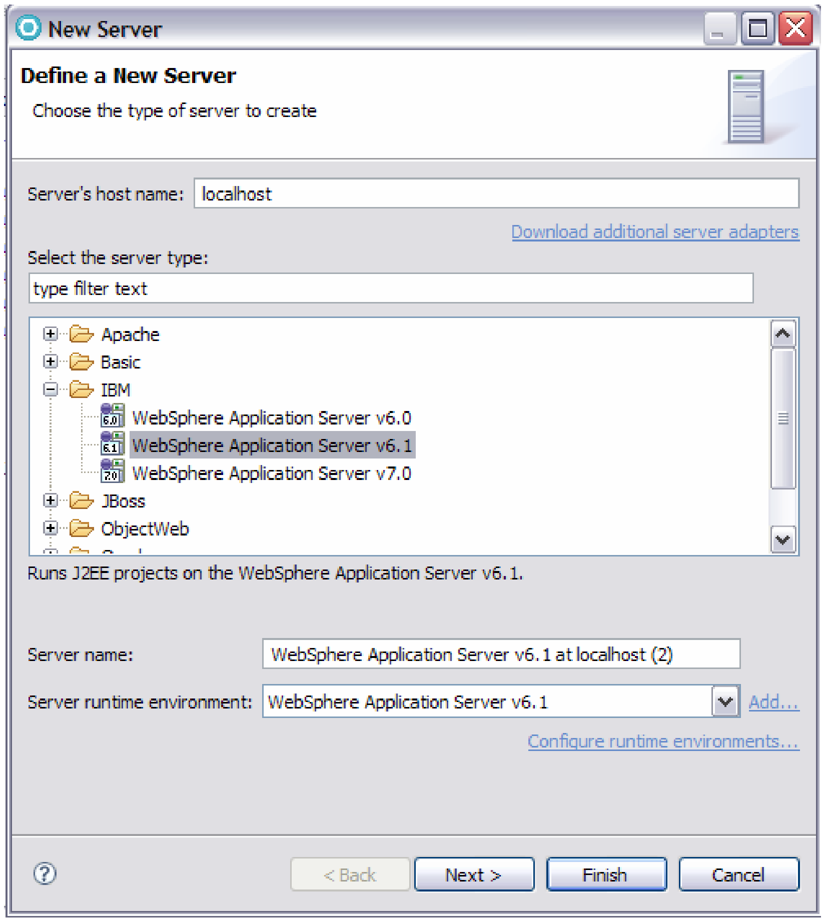
____ 5. **If a Server is not defined for your project yet**, add the WebSphere Application Server Version 6.1 If a WebSphere Application Server Version 6.1 is already defined skip to step 2.

__ a. In the lower panel, click the **Servers** tab.

__ b. Right-click in the lower panel and select **New > Server** from the contextual menu as shown below.



__ c. Select **IBM** > **WebSphere Application Server V6.1**


New Server
Apache
Basic
IBM
WebSphere Application Server v6.0
WebSphere Application Server v6.1
WebSphere Application Server v7.0
JBoss
ObjectWeb
< Back
Next >
Finish
Cancel

Define a New Server

Choose the type of server to create

Server's host name: localhost

Download additional server adapters

Select the server type:

type filter text

Runs J2EE projects on the WebSphere Application Server v6.1.

Server name: WebSphere Application Server v6.1 at localhost (2)

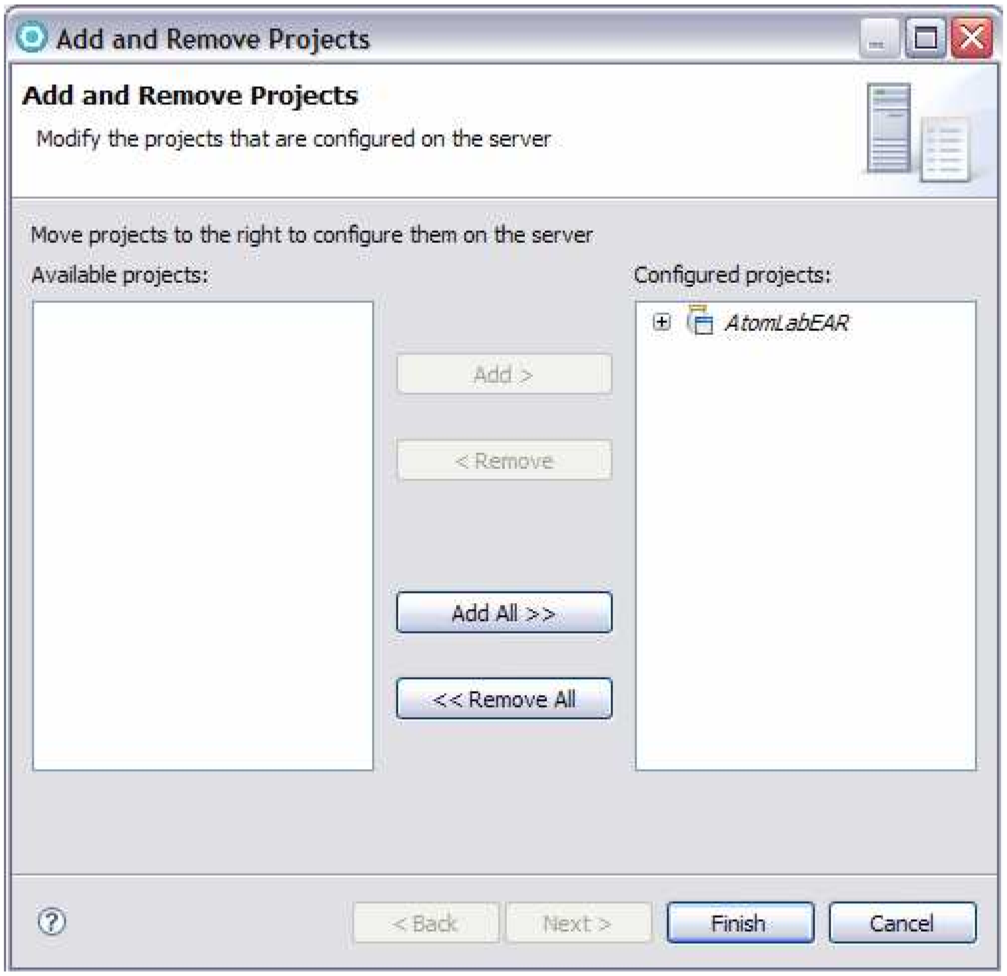
Server runtime environment: WebSphere Application Server v6.1 Add...

Configure runtime environments...

__ d. Click **Next**

__ e. Take the default settings and click **Next**

__ f. In the Add and Remove Projects select **AtomLabEAR** and click **Add**


Add and Remove Projects

Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

Available projects:

Configured projects:

AtomLabEAR

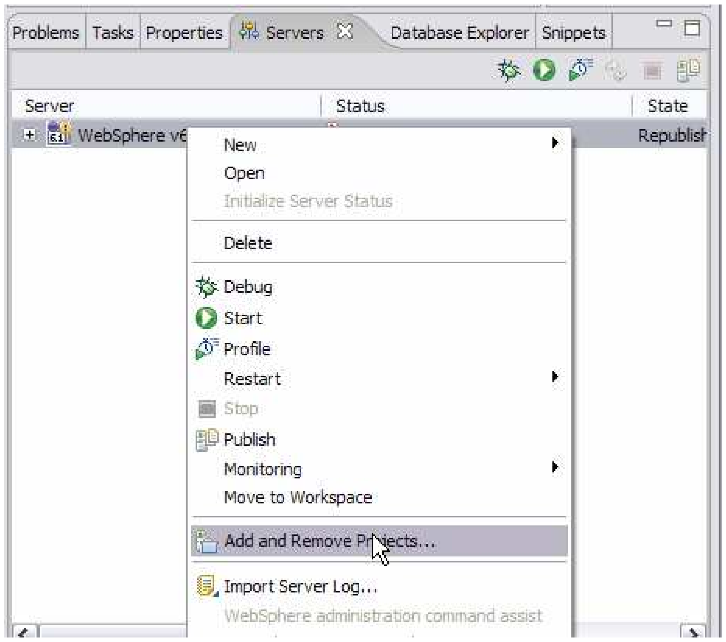
__ g. Click **Next**

__ h. Click **Finish**

____ 6. If you already have WebSphere Application Server V6.1 defined under Servers then just add the AtomLab Project to the Server.

__ a. In the lower panel, click the **Servers** tab.

__ b. Right-click WebSphere Application Server V6.1 in the lower panel and select **Add and Remote Projects** from the contextual menu as shown below.



__ c. Select **AtomLabEAR** and click **Add**

__ d. Click **Finish**

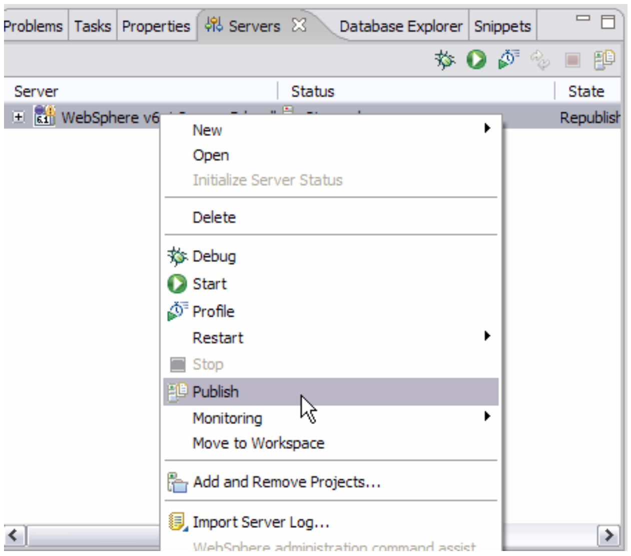
Note: this should be the only project associated with the server.

Part 5: Publish the AtomLab application

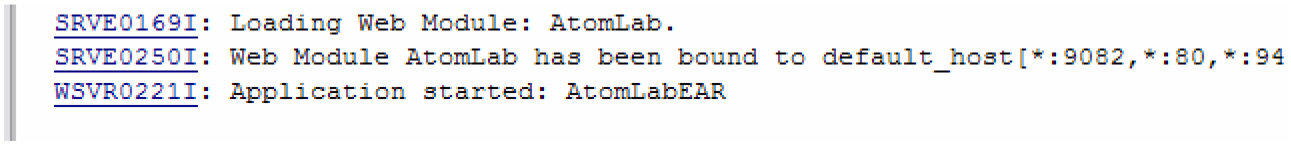
____ 7. Start the server and publish the AtomLab application

__ a. In the lower panel, click the **Servers** tab.

__ b. Right-click WebSphere Application Server V6.1 in the lower panel and select **Publish** from the contextual menu as shown below. The server will start and the AtomLab application will publish to the server too.



____ 8. In the console window you will notice that your AtomLab Application will be started. You can also view the console window to know the default port number.



```
SRVE0169I: Loading Web Module: AtomLab.
SRVE0250I: Web Module AtomLab has been bound to default_host[*:9082,*:80,*:94
WSVR0221I: Application started: AtomLabEAR
```

Part 6: Running the AtomLab application

____ 1. First you should view the two URLs you are gathering the Atom feed information from.

__ a. Open Internet Explorer and view the following URLS

1) http://news.google.com/?output=atom

2) http://www.oreillynet.com/pub/feed/1

__ b. Notice that in Internet Explorer you see xml code for the Atom feeds

__ c. Open Firefox and view the following URLS

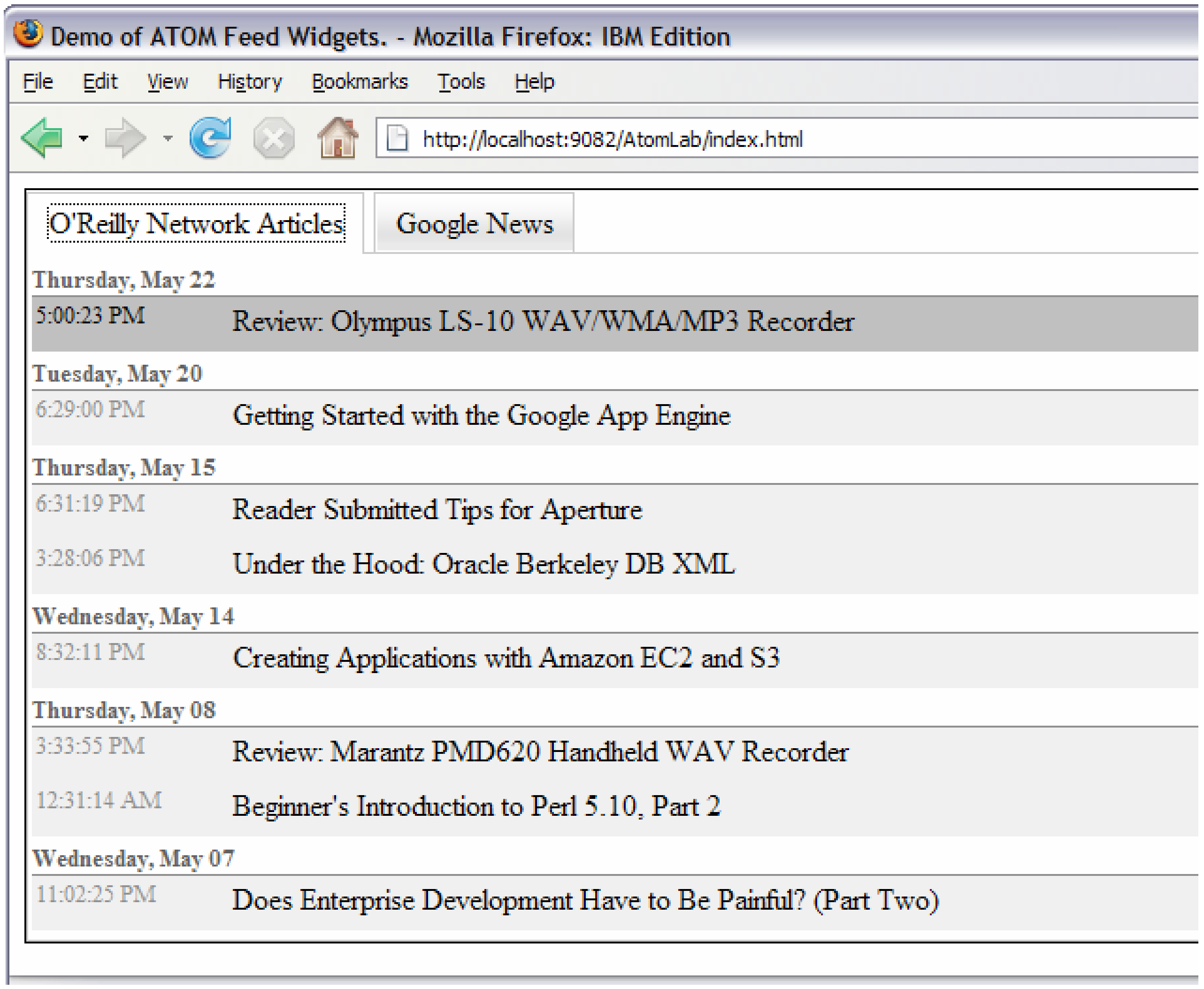
1) http://news.google.com/?output=atom

2) http://www.oreillynet.com/pub/feed/1

__ d. Notice that Firefox has the capabilities of having a friendly view of the Atom feeds

____ 2. Now go to your AtomDemo application

__ a. View the application by opening a **Firefox** browser to http://localhost:9082/AtomLab/index.html


Demo of ATOM Feed Widgets. - Mozilla Firefox: IBM Edition
File Edit View History Bookmarks Tools Help
http://localhost:9082/AtomLab/index.html
O'Reilly Network Articles
Google News

Thursday, May 22

5:00:23 PM	Review: Olympus LS-10 WAV/WMA/MP3 Recorder

Tuesday, May 20

6:29:00 PM	Getting Started with the Google App Engine

Thursday, May 15

6:31:19 PM	Reader Submitted Tips for Aperture
3:28:06 PM	Under the Hood: Oracle Berkeley DB XML

Wednesday, May 14

8:32:11 PM	Creating Applications with Amazon EC2 and S3

Thursday, May 08

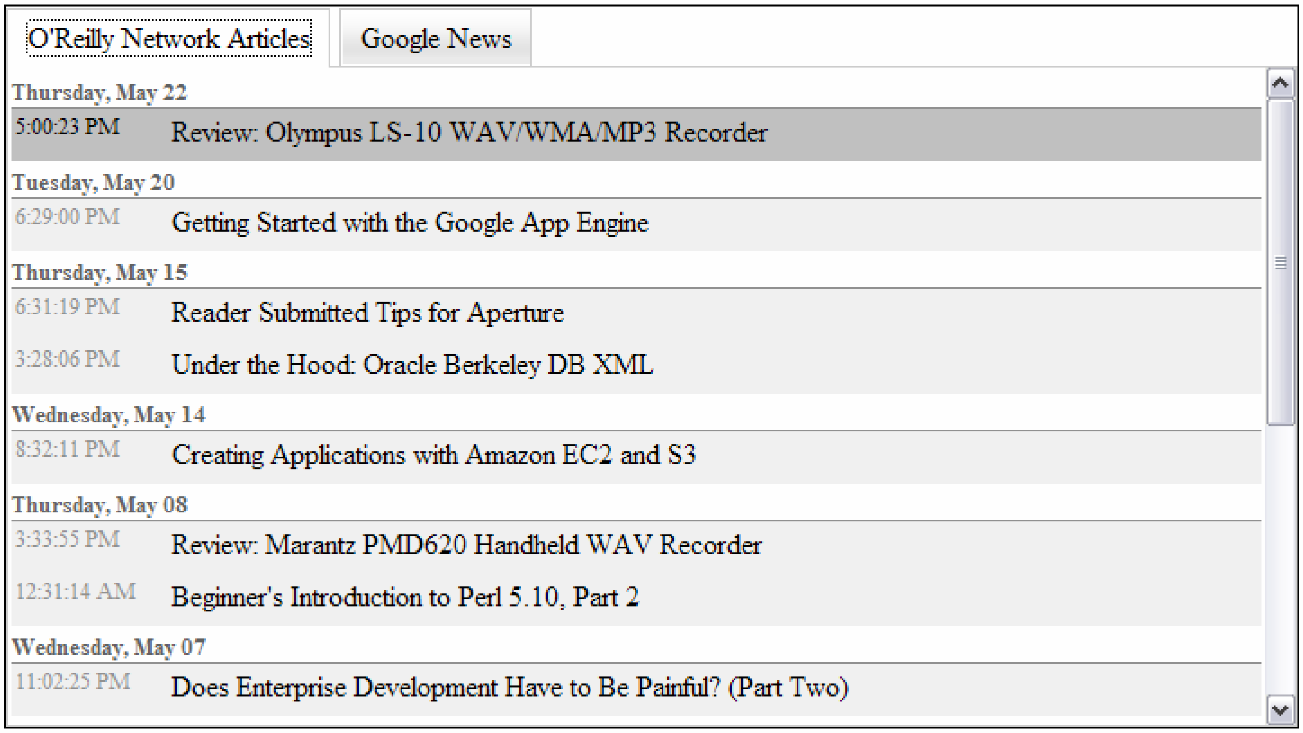
3:33:55 PM	Review: Marantz PMD620 Handheld WAV Recorder
12:31:14 AM	Beginner's Introduction to Perl 5.10, Part 2

Wednesday, May 07

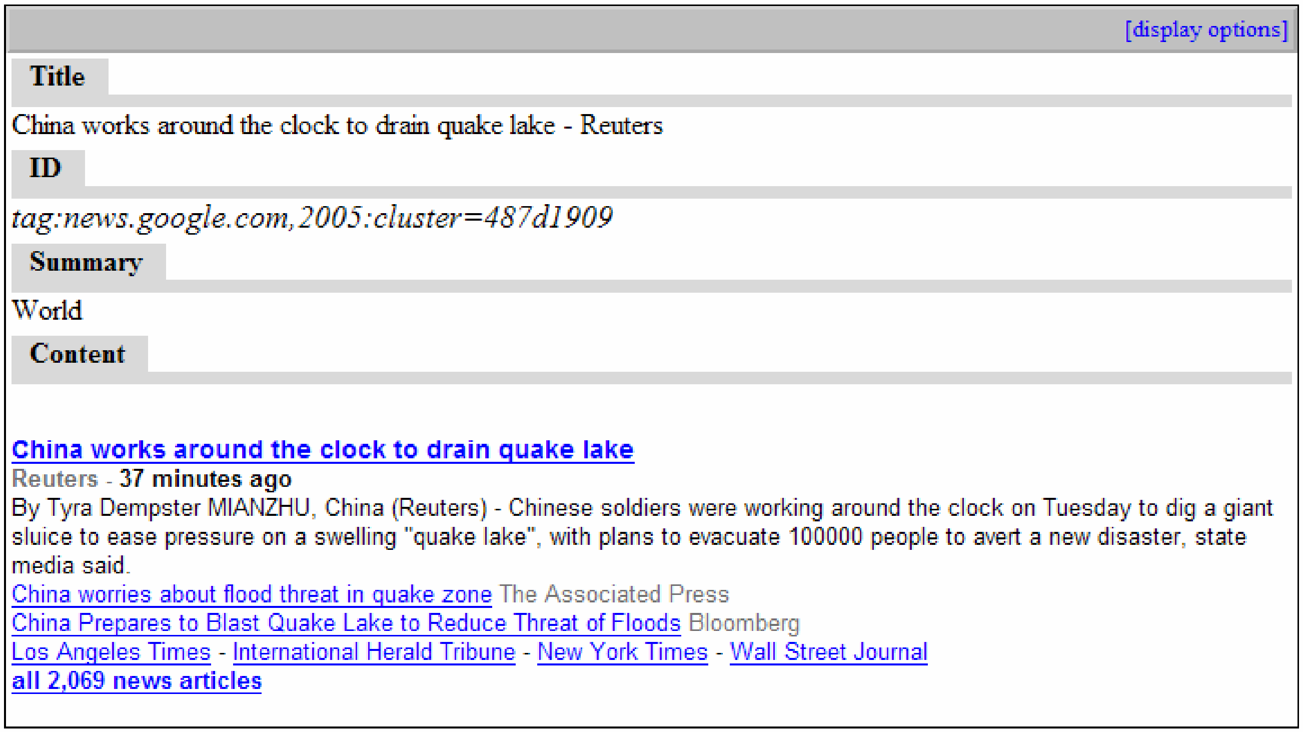
11:02:25 PM	Does Enterprise Development Have to Be Painful? (Part Two)

__ b. Notice for the O'Reilly Network Articles tab that the upper content is coming from the O'Reilly server and is being displayed by the FeedViewer widget. The upper content for the Google News tab is coming from the Google News server and is being displayed by the FeedViewer widget.

__ c. Click the O'Reilly Network Articles tab. Then click one of the Atom feeds. You will see the Atom feed data displayed on the lower part of the page. This information is coming from the O'Reilly server and is being displayed by the FeedEntryViewer Widget.



Thursday, May 22
5:00:23 PM Review: Olympus LS-10 WAV/WMA/MP3 Recorder
Tuesday, May 20
6:29:00 PM Getting Started with the Google App Engine
Thursday, May 15
6:31:19 PM Reader Submitted Tips for Aperture
3:28:06 PM Under the Hood: Oracle Berkeley DB XML
Wednesday, May 14
8:32:11 PM Creating Applications with Amazon EC2 and S3
Thursday, May 08
3:33:55 PM Review: Marantz PMD620 Handheld WAV Recorder
12:31:14 AM Beginner's Introduction to Perl 5.10, Part 2
Wednesday, May 07
11:02:25 PM Does Enterprise Development Have to Be Painful? (Part Two)



Title
China works around the clock to drain quake lake - Reuters
ID
tag:news.google.com,2005:cluster=487d1909
Summary
World
Content

China works around the clock to drain quake lake
Reuters - **37 minutes ago**
By Tyra Dempster MIANZHU, China (Reuters) - Chinese soldiers were working around the clock on Tuesday to dig a giant sluice to ease pressure on a swelling "quake lake", with plans to evacuate 100000 people to avert a new disaster, state media said.
China worries about flood threat in quake zone The Associated Press
China Prepares to Blast Quake Lake to Reduce Threat of Floods Bloomberg
Los Angeles Times - International Herald Tribune - New York Times - Wall Street Journal
all 2,069 news articles

__ d. Remember that the browser does not realize that the data is actually coming from two different servers as the application is using the Ajax proxy.

____ 3. Play around with the Atom Lab application. Notice how the Atom feed from the real URL gets displayed nicely in the Atom Lab application through the widgets.

What you did in this exercise

In this lab you created a sample application that used the Atom widgets and Ajax Proxy. The AtomLab application showed how to grab Atom feeds from two different sources and displayed them in a nice way using the Atom widget tools. The application used Ajax proxy to forward information to the requested server. From the client's perspective, the requests appear to originate from the same server, even though the Ajax proxy was forwarding requests to multiple Web servers.