Appendix A - Feeds Support

Migration from Apache Abdera to Apache Wink

Apache Wink is the perfect solution for consuming and producing Atom, APP and RSS documents. The following section describes how to migrate from Apache Abdera to Apache Wink by providing a set of examples that cover most use cases.

Advantages of Apache Wink over Apache Abdera

- Standardized APIs (using JAX-RS and JAXB)
- Support for handling XML and JSON more easily
- Support for handling RSS and ATOM more easily

This section contains the following topics:

---

1) Consuming Atom Documents

The following code example demonstrates the consumption of Atom documents using Apache Abdera.

Apache Abdera - Click on link to Download - ConsumeAtomUsingAbdera.java

```
Abdera abdera = new Abdera();
Parser parser = abdera.getParser();
URL url = new URL("http://alexharden.org/blog/atom.xml");
Document<Feed> doc = parser.parse(url.openStream());
Feed feed = doc.getRoot();
System.out.println(feed.getTitle());
for (Entry entry : feed.getEntries()) {
    System.out.println("\t" + entry.getTitle());
}
```

The following code example demonstrates the consumption of Atom documents using Apache Wink.

Apache Wink - Click on link to Download - ConsumeAtomUsingWink.java

```
RestClient client = new RestClient();
Resource resource = client.resource("http://alexharden.org/blog/atom.xml");
AtomFeed feed = resource.accept(MediaType.APPLICATION_ATOM_XML).get(AtomFeed.class);
System.out.println(feed.getTitle().getValue());
for (AtomEntry entry : feed.getEntries()) {
    System.out.println("\t" + entry.getTitle().getValue());
}
```

---

2) a) Producing Atom Documents

The following code example demonstrates the production of Atom documents using Apache Abdera.

Apache Abdera - Click on links to Download - ProduceAtomUsingAbdera.java ProduceAtomUsingAbdera_web.xml

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    Abdera abdera = new Abdera();
    Feed feed = abdera.newFeed();

    feed.setId("tag:example.org,2007:/foo");
    feed.setTitle("Test Feed");
    feed.setSubtitle("Feed subtitle");
    feed.setUpdated(new Date());
    feed.addAuthor("Shiva HR");
    feed.addLink("http://example.com");
    feed.addLink("http://example.com/foo", "self");

    Entry entry = feed.addEntry();
    entry.setId("tag:example.org,2007:/foo/entries/1");
    entry.setTitle("Entry title");
    entry.setSummaryAsHtml("<p>This is the entry title</p>");
    entry.setUpdated(new Date());
    entry.setPublished(new Date());
    entry.addLink("http://example.com/foo/entries/1");

    feed.getDocument().writeTo(response.getWriter());
}
```

The following code example demonstrates the production of Atom documents using Apache Wink.

Apache Wink - Click on links to Download - **ProduceAtomUsingWink.java** **ProduceAtomUsingWink_web.xml**

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    AtomFeed feed = new AtomFeed();
    feed.setId("tag:example.org,2007:/foo");
    feed.setTitle(new AtomText("Test Feed"));
    feed.setSubtitle(new AtomText("Feed subtitle"));
    feed.setUpdated(new Date());

    AtomPerson person = new AtomPerson();
    person.setName("Shiva HR");
    feed.getAuthors().add(person);

    AtomLink link1 = new AtomLink();
    link1.setHref("http://example.com");
    feed.getLinks().add(link1);

    AtomLink link2 = new AtomLink();
    link2.setHref("http://example.com/foo");
    link2.setRel("self");
    feed.getLinks().add(link2);

    AtomEntry entry = new AtomEntry();
    entry.setId("tag:example.org,2007:/foo/entries/1");
    entry.setTitle(new AtomText("Entry title"));

    AtomText summary = new AtomText();
    summary.setType(AtomTextType.html);
    summary.setValue("<p>This is the entry title</p>");
    entry.setSummary(summary);

    entry.setUpdated(new Date());
    entry.setPublished(new Date());

    AtomLink link3 = new AtomLink();
    link3.setHref("http://example.com/foo/entries/1");
    entry.getLinks().add(link3);

    feed.getEntries().add(entry);

    AtomFeed.marshal(feed, response.getOutputStream());
}
```

2) b) Producing Atom Documents - the JAX-RS way

A more elegant way of producing Atom documents using Apache Wink is the JAX-RS way as described below:

1. Open the Eclipse development environment and create a "Dynamic Web Project".
2. Add Apache Wink & its dependent JARs under Java EE Module Dependencies.
3. Create a POJO class and a method that creates Atom feed document. Annotate the class & its methods with the required JAX-RS annotations as below:
   ProduceAtom.java
4. Add org.apache.wink.server.internal.servlet.RestServlet into web.xml and specify the path of above Resource class in it's init-param.
   See ProduceAtomWinkElegant_web.xml and application
5. Deploy the web-application and access it using the url http://localhost:8080/ProduceAtom_Wink_Elegant/rest/getAtom
6. Final WAR -> ProduceAtom_Wink_Elegant.zip (add Wink & its dependent JARs under ProduceAtom_Wink_Elegant\WEB-INF\lib and re-zip it as WAR).

---

3) Consuming RSS Documents

The following code example demonstrates the consuming of RSS documents using Apache Abdera.

Apache Abdera - Click on link to Download - ConsumeRssUsingAbdera.java

```
public static void main(String[] args) throws ParseException, IOException {
    System.out.println("Consuming RSS Documents using Abdera...\n");
    Abdera abdera = new Abdera();
    Parser parser = abdera.getParser();
    URL url = new URL("http://www.rssboard.org/files/sample-rss-2.xml");
    Document<RssFeed> doc = parser.parse(url.openStream());
    RssFeed rssFeed = doc.getRoot();
    System.out.println("Title: " + rssFeed.getTitle());
    System.out.println("Description: " + rssFeed.getSubtitle() + "\n");
    int itemCount = 0;
    for (Entry entry : rssFeed.getEntries()) {
        System.out.println("Item " + ++itemCount + ":");
        System.out.println("\tTitle: " + entry.getTitle());
        System.out.println("\tPublish Date: " + entry.getPublished());
        System.out.println("\tDescription: " + entry.getContent());
    }
}
```

The following code example demonstrates the consuming of RSS documents using Apache Wink.

Apache Wink - Click on link to Download - ConsumeRssUsingWink.java

```
public static void main(String[] args) {
    System.out.println("Consuming RSS Documents using Apache Wink...\n");
    RestClient client = new RestClient();
    String url = "http://www.rssboard.org/files/sample-rss-2.xml";
    Resource resource = client.resource(url);
    RssFeed rss = resource.accept(MediaType.APPLICATION_XML).get(RssFeed.class);
    RssChannel channel = rss.getChannel();
    System.out.println("Title: " + channel.getTitle());
    System.out.println("Description: " + channel.getDescription() + "\n");
    int itemCount = 0;
    for (RssItem item : channel.getItems()) {
        System.out.println("Item " + ++itemCount + ":");
        System.out.println("\tTitle: " + item.getTitle());
        System.out.println("\tPublish Date: " + item.getPubDate());
        System.out.println("\tDescription: " + item.getDescription());
    }
}
```

4) Producing RSS Documents

Apache Abdera

Apache Abdera version 0.4 does not support RSS write.

Apache Wink

Same as in 2) b) Producing Atom Documents - the JAX-RS way. However the resource method now returns an RssFeed object instead of AtomFeed object.

Apache Wink - Click on link to Download - ProduceRss_Wink_Elegant.zip

```
@Path("/getRss")
public class ProduceRss {
    @GET
    @Produces(MediaType.APPLICATION_XML)
    public Rss getRss() {
        RssFeed rss = new RssFeed();

        RssChannel channel = new RssChannel();
        channel.setTitle("Liftoff News");
        channel.setLink("http://liftoff.msfc.nasa.gov");
        channel.setDescription("Liftoff to Space Exploration.");
        channel.setPubDate(new Date().toString());

        RssItem item = new RssItem();
        item.setTitle("Star City");
        item.setLink("http://liftoff.msfc.nasa.gov/news/2003/news-starcity.asp");
        item.setDescription("How do Americans get ready to work with Russians aboard the International Space
Station?");

        channel.getItem().add(item);
        rss.setChannel(channel);
        return rss;
    }
}
```

---

5) Writing Atom Publishing Protocol (APP) Server

The following steps explain how to implement an APP server as described in the following beautiful article by James Snell: http://www.ibm.com/developerworks/library/x-atompp1/

Apache Abdera

1. Open the Eclipse development environment and create a "Dynamic Web Project".
2. Add Apache Abdera & its dependent JARs under Java EE Module Dependencies.
3. Add the following CollectionAdapter and Provider classes under src/myPackage directory: APP_CollectionAdapter.java APP_ContentProvider.java
4. Add org.apache.abdera.protocol.server.servlet.AbderaServlet into web.xml and point the following init paramters to the classes added above.
   org.apache.abdera.protocol.server.Provider
   org.apache.abdera.protocol.server.CollectionAdapter
   APP_Server_Abdera_web.xml
5. Add the following index.jsp which has help on how to perform the APP operations: APP_Server_Abdera_index.jsp
6. Deploy and run the application.

Final WAR -> APP_Server_Abdera.zip (add Apache Abdera & its dependent JARs under APP_Server_Abdera\WEB-INF\lib and re-zip it as WAR).

Apache Wink

1. Open the Eclipse development environment and create a "Dynamic Web Project".
2. Add Apache Wink & its dependent JARs under Java EE Module Dependencies.
3. Add the following Resource class under src/myPackage directory: EntriesCollection.java
4. Add org.apache.wink.server.internal.servlet.RestServlet into web.xml and specify the path of above Resource class in it's init-param. APP_Server_Wink_web.xml APP_Server_Wink_application
5. Add the following index.jsp which has help on how to perform the APP operations: APP_Server_Wink_index.jsp
6. Deploy and run the application.

Final WAR -> APP_Server_Wink.zip (add Apache Wink & its dependent JARs under APP_Server_Wink\WEB-INF\lib and re-zip it as WAR)

References

- Apache Wink's "SimpleDefects" example: http://svn.apache.org/repos/asf/incubator/wink/tags/wink-0.1-incubating/wink-examples/apps/SimpleDefects/src/main/java/org/apache/wink/example/simpledefects/resources/DefectsResource.java
- Abdera Feed Sample shipped with IBM WebSphere Feature Pack for Web 2.0 http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.ajax.feed.samples.help/docs/GettingStarted_useage.html
- Abdera Server Implementation Guide -> http://cwiki.apache.org/ABDERA/server-implementation-guide.html
- Abdera Collection Adapter Implementation Guide -> http://cwiki.apache.org/ABDERA/collection-adapter-implementation-guide.html

---

6) Writing Atom Publishing Protocol (APP) Client

In order to write an Atom Publishing Protocol client refer to the following examples.

Important Note

Make sure that the APP_Server_Abdera.war and the APP_Server_Wink.war provided in the previous example are deployed before running these examples.

Apache Abdera - Click on link to Download - APP_Client_Abdera.java

1. Acessing Service Document:

```
Document<Service> introspection = abderaClient.get(SERVICE_URL).getDocument();
Service service = introspection.getRoot();
List<Workspace> workspaces = service.getWorkspaces();
for (Workspace workspace : workspaces) {
    System.out.println("\t" + workspace.getTitle());
    List<Collection> collections = workspace.getCollections();
    for (Collection collection : collections) {
        System.out.println("\t" + collection.getTitle() + "\t:\t" + collection.getHref());
    }
    System.out.print("\n");
}
```

2. Getting a Feed

```
RequestOptions opts = new RequestOptions();
opts.setContentType("application/atom+xml;type=feed");
ClientResponse response = abderaClient.get(FEED_URL, opts);
Feed feed = (Feed)response.getDocument().getRoot();
```

3. Posting an entry to a Feed

```
RequestOptions opts = new RequestOptions();
opts.setContentType("application/atom+xml;type=entry");
ClientResponse response = abderaClient.post(FEED_URL, newEntry, opts);
```

4. Putting a change to an Entry

```
RequestOptions opts = new RequestOptions();
opts.setContentType("application/atom+xml;type=entry");
ClientResponse response = abderaClient.put(ENTRY_URL, changedEntry.getDocument(), opts);
```

5. Getting an Entry

```
RequestOptions opts = new RequestOptions();
opts.setContentType("application/atom+xml;type=entry");
ClientResponse response = abderaClient.get(ENTRY_URL, opts);
Entry entry = (Entry)response.getDocument().getRoot();
```

6. Deleting an Entry

```
ClientResponse response = abderaClient.delete(ENTRY_URL);
```

Apache Wink - Click on link to Download - APP_Client_Wink.java

1. Acessing Service Document:

```
Resource resource = restClient.resource(SERVICE_URL);
AppService service = resource.accept(MediaTypeUtils.ATOM_SERVICE_DOCUMENT).get(AppService.class);
List<AppWorkspace> workspaces = service.getWorkspace();
for (AppWorkspace workspace : workspaces) {
    System.out.println("\t" + workspace.getTitle().getValue());
    List<AppCollection> collections = workspace.getCollection();
    for (AppCollection collection : collections) {
        System.out.println("\t" + collection.getTitle().getValue()
            + "\t:\t"
            + collection.getHref());
    }
    System.out.print("\n");
}
```

2. Getting a Feed

```
Resource feedResource = restClient.resource(FEED_URL);
AtomFeed feed = feedResource.accept(MediaType.APPLICATION_ATOM_XML).get(AtomFeed.class);
```

3. Posting an entry to a Feed

```
Resource feedResource = restClient.resource(FEED_URL);
ClientResponse response =
    feedResource.contentType(MediaType.APPLICATION_ATOM_XML).post(newEntry);
```

4. Putting a change to an Entry

```
Resource feedResource = restClient.resource(ENTRY_URL);
ClientResponse response =
    feedResource.contentType(MediaType.APPLICATION_ATOM_XML).put(changedEntry);
```

5. Getting an Entry

```
Resource feedResource = restClient.resource(ENTRY_URL);
AtomEntry atomEntry = feedResource.accept(MediaType.APPLICATION_ATOM_XML).get(AtomEntry.class);
```

6. Deleting an Entry

```
Resource feedResource = restClient.resource(ENTRY_URL);
ClientResponse response = feedResource.delete();
```