Atomized

How to consume and publish Atom using Open-Source Java tools

Ugo Cei
Sourcesense
u.cei@sourcesense.com

sourcesense

What is Atom

- Atom is a syndication format
  - RFC 4287
- Atom is a publishing protocol
  - draft-ietf-atompub-protocol-11.txt

Atom as a Syndication Format

- Analogous to RSS but arguably "better", as in:
- Less ambiguous
- Richer

Problems with RSS 2.0

Hat tip: Dave Johnson

- Spec is too loose and unclear:
  - What fields can be escaped HTML?
  - How many enclosures are allowed per element?
- Content model is weak:
  - No support for summary and content.
  - Content-type and escaping not specified.
- RSS Board not allowed to clarify specification.

Atom Content Types

- Plain text

```
<content type=”text”>Some Text</content>
```

- Escaped

```
<content type=”html”>
  &lt;div&gt;...&lt;/div&gt;
</content>
```

- Well-formed

```
<content type=”xhtml”>
  <div>...</div>
</content>
```

Atom Content Types

- XML

```
<content type=”application/rdf+xml”>
  <rdf:RDF>...</rdf:RDF>
</content>
```

- External

```
<content type=”application/xyz”
  src=”http://example.com/whatever”/>
```

Atom as a Publishing Protocol

- "Application-level protocol for publishing and editing Web resources using HTTP"
- Based on Atom Syndication Format.
- Began as a replacement for the old XML-RPC based blog APIs.
- Embodiment of the REST principles.

Atom Publishing Protocol

```
GET /entries HTTP/1.1

200 OK
Content-Type: application/atom+xml

<a:feed>
  <a:entry>
    <a:link rel='self' href='/entries/1' />
    ...
  </a:entry>
  ...
</a:feed>
```

Atom Publishing Protocol

```
GET /entries/1 HTTP/1.1

200 OK
Content-Type: application/atom+xml


<a:entry>
  <a:link rel='self' href='/entries/1' />
  <a:link rel='edit'
      href='/entries/1;edit' />
  ...
</a:entry>
```

Atom Publishing Protocol

```
POST /entries HTTP/1.1
Content-Type: application/atom+xml

<a:entry>
   <a:title>An entry</a:title>
   ...
</a:entry>
```

```
201 Created
Location: http://example.com/entries/2
```

Atom Publishing Protocol

```
PUT /entries/2;edit HTTP/1.1
Content-Type: application/atom+xml

<a:entry>
  <a:title>Updated entry</a:title>
  ...
</a:entry>

200 OK
<a:entry>
  ...
</a:entry>
```

Atom Publishing Protocol

```
DELETE /entries/1 HTTP/1.1

200 OK
```

The Service Document

```
<service>
  <workspace>
    <atom:title>Main Site</atom:title>
    <collection href="http://e.org/entries">
      <atom:title>Blog Entries</atom:title>
      <accept>entry</accept>
    </collection>
    <collection href="http://e.org/pix">
      <atom:title>My Pictures</atom:title>
      <accept>image/*</accept>
    </collection>
  </workspace>
</service>
```

“Beyond Blogging”

Hat tip: Dave Johnson

- Blogger supports Atom and APP. Wordpress does it with a plugin, but there’s more.
- A new foundation for REST based web services:
  - Google DATA API.
  - Lucene Web Services.
  - Ning.

Apache Abdera

- "The goal of the Apache Abdera project is to build a functionally-complete, high-performance implementation of the Atom Syndication Format and Atom Publishing Protocol specifications."

incubator.apache.org/abdera

Parsing a Feed

```
Parser parser = Abdera.getNewParser();
URI uri = new URI("http://example.org/feed.xml");
InputStream in = uri.toURL().openStream();
Document<Feed> doc = parser.parse(in, uri);
Feed feed = doc.getRoot();
```

Note: These samples use the (unreleased) 0.2.0 APIs.
Using 0.1.0, you'd have to get a Parser by:

```
Parser parser = Parser.INSTANCE;
```

Elements of a Feed

```
feed.getAlternateLink();
feed.getAuthors();
feed.getCategories();
feed.getContributors();
feed.getGenerator();
feed.getIcon();
feed.getId();
feed.getLinks();
feed.getLogo();
feed.getRights();
feed.getSubtitle();
feed.getTitle();
feed.getUpdated();

feed.getEntries();
```

StAX

- Abdera's parser implementation is based on the Streaming API for XML (JSR173).
- This allows it to use very little memory compared to DOM.
- Be careful not to close a stream too early! (Using *AutoReleasingInputStream* could help.)

Configuring the Parser

```
URI uri = new URI("http://example.org/feed.xml");
InputStream in = uri.toURL().openStream();
Parser parser = Abdera.getNewParser();
ParserOptions options = parser.getDefaultParserOptions();
options.setCharset("utf-8");
//.. set other parser options
Document<Feed> doc = parser.parse(in, uri, options);
```

Creating a Feed Document

```
Factory factory = Abdera.getNewFactory();
Feed feed = factory.newFeed();
feed.setId("tag:example.org,2005:/myfeed", false);
feed.setTitle("My Example Feed");
// .. set other feed properties
Document<Feed> doc = feed.getDocument();
doc.writeTo(System.out);
```

Using Extensions

```
Factory factory = Abdera.getNewFactory();
Feed feed = factory.newFeed();
// ... set other feed properties
feed.addSimpleExtension(
  new QName("urn:foo", "myExtension", "a"),
    "This is an extension");

Link link = feed.addLink("http://example.org");
link.setAttributeValue(
  new QName("urn:foo", "myAttribute", "a"),
    "My Attribute");
```

Using Extensions

```
<?xml version='1.0' ?>
<feed xmlns='http://www.w3.org/2005/Atom'>
  ...
  <a:myExtension xmlns:a="urn:foo">
    This is an extension
  </a:myExtension>
  <link href="http://example.org"
        xmlns:a="urn:foo"
        a:myAttribute="My Attribute" />
</feed>
```

Atom Protocol Support

Client

- The *Client* interface is the basic abstraction.
- As of now, the only implementation provided is *CommonsClient*, which uses Jakarta Commons HttpClient.

APP: Reading the Service Document

```
Client client = new CommonsClient();
RequestOptions options = client.getDefaultRequestOptions();
options.setHeader("Connection", "close");
// do the introspection step
ClientResponse response =
    client.get("http://localhost:8080/service",options);
Document<Service> serviceDoc = response.getDocument();
Workspace workspace = serviceDoc.getRoot().getWorkspace("Test");
```

APP: Posting and Editing an Entry

```
Entry entry = factory.newEntry();
// ... set entry properties
response = client.post(colUri, entry, options);
// read the new entry
String selfUri = response.getLocation().toString();
response = client.get(selfUri, options);
Document<Entry> doc = response.getDocument();
// get the edit uri from the entry
String editUri = doc.getRoot().getEditLink().getHref().toString();
// change the entry
entry = (Entry) doc.getRoot().clone();
entry.setTitle("New title");
// submit the changed entry back to the server
response = client.put(editUri, entry, options);
```

Extras

- XPath
- Digital Signatures
- Encryption
- Filtering
- IRI Support (RFC 3987)
- JSON
- Atom Threading Extensions (RFC 4685)

Using XPath

```
Document<Feed> doc = parser.parse(inputStream, uri);
XPath xpath = Abdera.getXPath();
// Select the id of the document
String id = xpath.valueOf("/a:feed/a:id", doc);
// Select all entries from the document
List entries = xpath.valueOf("//a:entry", doc);
for (Iterator i = entries.iterator(); i.hasNext();) {
  Entry entry = (Entry)i.next();
  //...
}
// Determine if a feed contains a specific extension
boolean hasFoo = xpath.isTrue("//x:foo", doc);
// The XPath support works on any element in the FOM
Entry entry = (Entry) xpath.selectSingleNode("//a:entry", doc);
String id = xpath.valueOf("a:id", entry);
```

Digitally Signing an Atom Document

```
Factory factory = Abdera.getNewFactory();
Feed feed = factory.newFeed();
PrivateKey myPrivateKey = ...
X509Certificate myX509Cert = ...
Signature sig = new AbderaSecurity().getSignature();
SignatureOptions options = sig.getDefaultSignatureOptions();
options.setSigningKey(myPrivateKey);
options.setCertificate(myX509Cert);
feed = sig.sign(feed, options);
//any modifications to the feed after this point will break the
signature
```

Encrypting an Atom Document

```
Feed feed = Abdera.getNewFeed();
Key kek = ... // Key encryption key
Key dek = ... // Data encryption key
Encryption enc = new AbderaSecurity().getEncryption();
EncryptionOptions options = enc.getDefaultEncryptionOptions();
options.setKeyEncryptionKey(kek);
options.setDataEncryptionKey(dek);
options.setIncludeKeyInfo(true);
Document doc = enc.encrypt(feed.getDocument(), options);
doc.writeTo(System.out); // output the encrypted XML
```

ParseFilter

```
/**
 * ParseFilter's determine which elements and attributes are
 * acceptable within a parsed document. They are set via the
 * ParserOptions.setParseFilter method.
 */
public interface ParseFilter extends Cloneable {
  public boolean acceptable(QName qname);
  public boolean acceptable(QName qname, QName attribute);
  ...
}
```

TextFilter

```
/**
 * Text filter provides a means of filtering unacceptable text
 * from elements in the Atom feed, including unwanted text and
 * markup in HTML entries.
 */
public interface TextFilter extends Cloneable {

  /**
   * Applies the text filter to the specified attribute text,
   * returns the filtered text
   */
  public String applyFilter(
    String value,
    Element parent,
    QName attribute);
```

IRIs

http://www.詹姆斯.com/feed

Threading

```
Factory factory = Abdera.getNewFactory();
Entry entry1 = factory.newEntry();
entry1.setId("tag:example.org,2006:/original");
// set other properties
Entry entry2 = Factory.INSTANCE.newEntry();
entry2.setId("tag:example.org,2006:/response");
// set other properties
ThreadHelper.addInReplyTo(entry2, entry1);
```

Threading

```
Factory factory = Abdera.getNewFactory();
Feed feed = factory.newFeed();
Link replies = feed.addLink("http://example.org/feed/comments",
  Link.REL_REPLIES);
// optionally set the reply count and last updated timestamp
replies.setAttributeValue(Constants.THRCOUNT, "10");
replies.setAttributeValue(Constants.THRUPDATED,
  AtomDate.format(new Date()));
```

ROME

- "ROME is an open source (Apache license) set of Atom/RSS Java utilities that make it easy to work in Java with most syndication formats: RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, and Atom 1.0."

rome.dev.java.net

Reading a Feed

```
SyndFeedInput input = new SyndFeedInput();
SyndFeed feed = input.build(new XmlReader(feedUrl));
```

With caching:

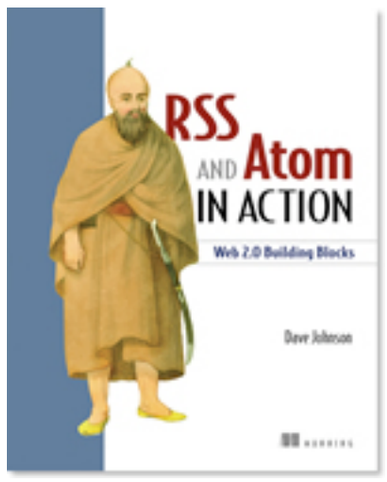
```
FeedFetcherCache feedInfoCache = HashMapFeedInfoCache.getInstance();
FeedFetcher feedFetcher = new HttpURLFeedFetcher(feedInfoCache);
SyndFeed feed = feedFetcher.retrieveFeed(feedUrl);
```

Converting a Feed

```
SyndFeedInput input = new SyndFeedInput();
SyndFeed feed = input.build(new XmlReader(feedUrl));
// outputType can be one of rss_0.9, rss_0.91, rss_0.92,
// rss_0.93,rss_0.94, rss_1.0, rss_2.0, atom_0.3, atom_1.0
feed.setFeedType(outputType);
SyndFeedOutput output = new SyndFeedOutput();
```

Blogapps

- "This project hosts the examples and utilities from *RSS and Atom In Action* by Dave Johnson. These examples and utilities are designed to be useful even if you haven't read the book."



blogapps.dev.java.net

Blogapps

- Blogapps provides a level of abstraction over APP and the MetaWeblog API:

```
public interface Blog ...

public class AtomBlog implements Blog ...

public class MetaWeblogBlog implements
    Blog ...
```

Roller

- “Roller is the open source blog server that drives Sun Microsystem's blogs.sun.com employee blogging site, IBM DeveloperWorks blogs, thousands of internal blogs at IBM Blog Central, the Javalobby's 10,000 user strong JRoller Java community site, and hundreds of other blogs world-wide.”

incubator.apache.org/projects/roller.html

TailRank FeedParser

- “Our Java-based FeedParser started off as the Jakarta FeedParser which was originally contributed to the ASF by Rojo. [...] Now that we don't have to yield to the ASF's 40 step release process we should have a 1.0 release soon.”

Sat Jan 14 2006

Tailrank

tailrank.com/code.php