Extensions

Abdera Extensions

Main Extensions

Bidi

The Atom Bidi Extension is a work in progress described by an IETF Internet-Draft. It is used to specify the base directionality of bidirection text within an Atom document.

Using the bidi attribute in an Atom document

```
<entry xmlns="http://www.w3.org/2005/Atom" dir="rtl">
  <title>W3C? (World Wide Web Consortium)????? ?? ?????? ????? ??????? ? - ERCIM.</title>
</entry>
```

When this text is displayed, the Unicode Bidirectional Algorithm needs to be used to display the text correctly. Unfortunately, Atom does not included a way of specifying the base directionality of text, which is why the bidi
attribute was created. See here for more information.

Setting the text direction using BidiHelper

```
Entry entry = abdera.newEntry();
BidiHelper.setDirection(BidiHelper.Direction.RTL, entry);
entry.setTitle("W3C? (World Wide Web Consortium)????? ?? ?????? ????? ??????? ? - ERCIM.");
```

The BidiHelper class can then be used to properly render the text:

Properly rendering the bidi text in the title

```
System.out.println(BidiHelper.getBidiElementText(entry.getTitleElement()));
```

The BidiHelper class can also be used to discover the base directionality of text

Getting the text direction

```
BidiHelper.Direction dir = BidiHelper.getDirection(entry);
```

If the dir attribute has not been set, the direction may be determined by applying a variety of guessing algorithms. Such algorithms are inherently flawed in that there is no reliable means of always guessing the right direction in every case.

Guessing the text direction

```
dir = BidiHelper.guessDirectionFromLanguage(entry);
dir = BidiHelper.guessDirectionFromTextProperties(entry);
dir = BidiHelper.guessDirectionFromJavaBidi(entry);
dir = BidiHelper.guessDirectionFromEncoding(entry);
```

When guessing the text direction based on the xml:lang value, the following rules apply:

- xml:lang values whose primary language tag is "ar","fa","ur","ps","syr","dv","he", or "yi" will always be Right-To-Left
- xml:lang values whose script tag is "arab","avst","hebr","hung","lydi","mand", "mani","mero","mong","nkoo","orkh","phlv", "phnx","samr","syrc","syre","syrj","syrn", "tfng", or "thaa" will always be Right-To-Left
- All other values are considered to have unspecified direction

When guessing the text direction based on the text properties, text strings that consist primarily of Right-To-Left characters will be assumed to be Right-To-Left.

When guessing the text direction based on the character encoding, the following encodings will always be considered as Right-To-Left: "iso-8859-6", "iso-8859-6-bidi", "iso-8859-6-i", "iso-ir-127", "ecma-114", "asmo-708", "arabic", "csisolatinarabic", "windows-1256", "ibm-864", "macarabic", "macfarsi", "iso-8859-8-i", "iso-8859-8-bidi", "windows-1255", "iso-8859-8", "ibm-862", "machebrew", "asmo-449", "iso-9036", "arabic7", "iso-ir-89", "csiso89asmo449", "iso-unicode-ibm-1264", "csunicodeibm1264", "iso_8859-8:1988", "iso-ir-138", "hebrew", "csisolatinhebrew", "iso-unicode-ibm-1265", "csunicodeibm1265", "cp862", "862", "cspc862latinhebrew"

Features

The Atompub Features draft is a work-in-progress extension to the Atom Publishing Protocol Service document format that is used to describe the characteristics and behaviors implemented by a server implementation.

The API for working with the features extension is a work-in-progress.

Setting and getting the features

```
Service service = abdera.newService();
Workspace workspace = service.addWorkspace("test");
Collection collection = workspace.addCollection("foo", "href");

Features features = FeaturesHelper.addFeaturesElement(collection);
features.addFeatures(
  FeaturesHelper.FEATURE_SUPPORTS_DRAFTS,
  FeaturesHelper.FEATURE_SUPPORTS_BIDI,
  FeaturesHelper.FEATURE_REQUIRES_XHTML_TEXT
);

FeaturesHelper.getFeatureStatus(collection, FeaturesHelper.FEATURE_SUPPORTS_BIDI);
FeaturesHelper.getFeatureStatus(collection, FeaturesHelper.FEATURE_SUPPORTS_GEO);
```

Paging and Archiving

The Feed Paging and Archiving Extensions are defined by RFC 5005.

Setting feed paging links

```
Feed feed = abdera.newFeed();

FeedPagingHelper.setCurrent(feed, "current");
FeedPagingHelper.setNext(feed, "next");
FeedPagingHelper.setPrevious(feed, "previous");
FeedPagingHelper.setFirst(feed, "first");
FeedPagingHelper.setLast(feed, "last");
FeedPagingHelper.setNextArchive(feed, "next-archive");
FeedPagingHelper.setPreviousArchive(feed, "prev-archive");
```

Result

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <link rel="current" href="current" />
  <link rel="next" href="next" />
  <link rel="previous" href="previous" />
  <link rel="first" href="first" />
  <link rel="last" href="last" />
  <link rel="next-archive" href="next-archive" />
  <link rel="prev-archive" href="prev-archive" />
</feed>
```

The FeedPagingHelper class can also be used to indicate whether a feed is a "complete" or "archive" feed

Result

```
FeedPagingHelper.setComplete(feed,true);
FeedPagingHelper.setArchive(feed,true);

boolean c = FeedPagingHelper.isComplete(feed);
boolean a = FeedPagingHelper.isArchive(feed);
```

License Extensions

The Atom License Extension is defined by RFC 4946.

Adding and getting license links from an entry

```
LicenseHelper.addLicense(entry, "http://creativecommons.org/licenses/by-nc/2.5/rdf");

List<Link> licenses = LicenseHelper.getLicense(entry);
```

Threading Extensions

The Atom Threading Extensions are defined by RFC 4685

Adding a thr:in-reply-to element to an entry

```
// this entry is a reply to entry "http://example.org/entries/1")
ThreadHelper.addInReplyTo(entry, "http://example.org/entries/1");

List<InReplyTo> irt = ThreadHelper.getInReplyTo(entry);
```

GeoRSS Extensions

The GeoRSS extension allows geospatial data to be added to Atom entries.

Adding geospatial data to an entry

```
Entry entry = abdera.newEntry();
Point point = new Point(36.331445,-119.64592);
GeoHelper.addPosition(entry, point);
```

Result

```
<entry xmlns="http://www.w3.org/2005/Atom"
       xmlns:georss="http://www.georss.org/georss/10">
  <georss:point>36.331445 -119.64592</georss:point>
</entry>
```

Points, Lines and Polygons can be added to an entry.

Retrieving geospatial data from an entry

```
Position[] positions = GeoHelper.getPositions(entry);
```

OpenSearch Extensions

Abdera supports a subset of the OpenSearch extensions

Adding OpenSearch elements to a feed

```
Feed feed = abdera.newFeed();
IntegerElement ie = feed.addExtension(OpenSearchConstants.ITEMS_PER_PAGE);
ie.setValue(10);

ie = feed.addExtension(OpenSearchConstants.START_INDEX);
ie.setValue(0);

ie = feed.addExtension(OpenSearchConstants.TOTAL_RESULTS);
ie.setValue(100);
```

Result

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:os="http://a9.com/-/spec/opensearch/1.1/">
  <os:itemsPerPage>10</os:itemsPerPage>
  <os:startIndex>0</os:startIndex>
  <os:totalResults>100</os:totalResults>
</feed>
```

MediaRSS Extensions

Abdera supports an experimental implementation of the MediaRSS extensions that can be used to add extended media metadata to an entry.

Adding MediaRSS elements to a feed

```
MediaTitle mtitle = entry.addExtension(MediaConstants.TITLE);
mtitle.setText("Media title");

MediaRestriction mrestriction = entry.addExtension(MediaConstants.RESTRICTION);
mrestriction.setType(RestrictionType.COUNTRY);
mrestriction.setRelationship(Relationship.ALLOW);
mrestriction.setText("au us");
```

Result

```
<entry xmlns="http://www.w3.org/2005/Atom"
       xmlns:media="http://search.yahoo.com/mrss/">
  <media:title>Media title</media:title>
  <media:restriction type="country" relationship="allow">au us</media:restriction>
</entry>
```

Auth Extenions

GoogleLogin

Abdera provides a limited implementation of the GooleLogin authentication scheme used by GData.

GData Authentication

```
GoogleLoginAuthScheme.register();
AbderaClient client = new CommonsClient();
client.addCredentials(
  "http://beta.blogger.com",
  null, "GoogleLogin",
  new UsernamePasswordCredentials("email","password")
);
```

The example above has the downside of performing the GoogleLogin authentication on each request. The code below performs the authentication once and reuses it for each subsequent request.

GData Authentication

```
GoogleLoginAuthCredentials credentials =
  new GoogleLoginAuthCredentials("email", "password", "blogger");
client.addCredentials(
  "http://beta.blogger.com",
  null, null, credentials
);
```

WSSE Login

Abdera also provides an implementation of the WSSE authentication mechanism

WSSE Authentication

```
AbderaClient client = new AbderaClient();
WSSEAuthScheme.register(client,true);
client.addCredentials(
  "http://example.org",
  null, "WSSE",
  new UsernamePasswordCredentials("email","password")
```

OAuth

Abdera also provides a client implementation of the OAuth authorization scheme

OAuth Authentication

```
AbderaClient client = new AbderaClient();
OAuthScheme.register(client, true);
OAuthCredentials credentials = new OAuthCredentials("consumer_key", "token", "signature_method", "realm");
client.addCredentials("http://example.org", null, "OAuth", credentials);
```

Simple Sharing Extensions

Abdera supports an experimental implementation of the Simple Sharing Extensions. The following examples shows a minimal example of using SSE to merge entries from one feed into another.

Merging two SSE Feeds

```
Feed f1 = abdera.newFeed();
f1.newId();
Entry e1 = f1.addEntry();
e1.newId();
e1.setTitle("Test");
SharingHelper.getSharing(f1,true);
Sync sync = SharingHelper.getSync(e1,true);
sync.setId(e1.getId().toString());

Feed f2 = abdera.newFeed();
f2.newId();
SharingHelper.getSharing(f2,true);
Entry e2 = f2.addEntry();
e2.newId();
sync = SharingHelper.getSync(e2,true);

SharingHelper.mergeFeeds(f1, f2);

System.out.println(f2);
```

Result

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:sx="http://www.microsoft.com/schemas/sse">
  <id>urn:uuid:83BD1E87EE96454DF21193779431187</id>
  <sx:sharing />
  <entry>
    <id>urn:uuid:83BD1E87EE96454DF21193779431228</id>
    <sx:sync />
  </entry>
  <entry>
    <id>urn:uuid:83BD1E87EE96454DF21193779431115</id>
    <title type="text">Test</title>
    <sx:sync id="urn:uuid:83BD1E87EE96454DF21193779431115" />
  </entry>
</feed>
```

The Sharing Extensions implementation also provides a means of resolving merge conflicts.

Resolving conflicts

```
ConflictResolver resolver =
  new ConflictResolver() {
    public Entry resolve(Entry entry, List<Entry> conflicts) {
      // resolve the conflicts in the entry
      return entry;
    }
};
for (Entry entry : f2.getEntries()) {
  if (SharingHelper.hasConflicts(entry)) {
    Entry resolved = SharingHelper.resolveConflicts(entry, resolver, "jms");
  }
}
```

JSON Writer

Abdera supports the ability to serialize Abdera objects out to a JSON representation. The JSON representation is documented here.

Using the JSON Writer

```
Entry entry = abdera.newEntry();
entry.setTitle("Title");
entry.setRightsAsHtml("<p>Copyright &copy; 2007</p>");
entry.newId();
entry.addAuthor("John Doe");
entry.setContentAsXhtml("<p>This is a <b>test</b></p>");

Writer json = abdera.getWriterFactory().getWriter("json");
entry.writeTo(json, System.out);
```

Result


```
{
 "id":"urn:uuid:5B32E821725995E5DE1193780060026",
 "title":"Title",
 "rights":{
  "attributes":{
   "type":"html"
  },
  "children":[{
    "name":"p",
    "attributes":{},
    "children":["Copyright \u00a9 2007"]
   }
  ]
 },
 "content":{
  "attributes":{
   "type":"xhtml"
  },
  "children":[{
    "name":"p",
    "attributes":{},
    "children":[
      "This is a ",
      {
        "name":"b",
        "attributes":{},
        "children":["test"]
      }
    ]
   }
  ]
 },
 "authors":[{
   "name":"John Doe"
  }
 ]
}
```