Atompub Working Group   R. Sayre
Internet-Draft   January 24, 2006
Expires: July 28, 2006

The Atom Publishing Protocol (Basic)
draft-sayre-atompub-protocol-basic-06

Status of this Memo

Copyright Notice

Abstract

This memo presents a protocol that uses XML and HTTP to publish and edit Web resources.

Table of Contents

1. Introduction

The Atom Publishing Protocol uses HTTP [RFC2616] and XML [W3C.REC-xml-20040204] to publish and edit Web resources.

2. Notational Conventions

The Atom Protocol namespace is "http://purl.org/atom/app#". This specification refers to it by using the prefix "pub", but that prefix is arbitrary.

The terms 'URI' and 'IRI' are shorthand for the identifiers specified in [RFC3986] and [RFC3987].

3. The Atom Publishing Protocol Model

The Atom Protocol uses HTTP to operate on collections of Web resources represented by Atom feeds [AtomFormat]. This section illustrates the editing cycle for Atom entries.

- o GET is used to retrieve a representation of a resource or perform a read-only query.
- o POST is used to create a new, dynamically-named resource.
- o PUT is used to update a known resource.
- o DELETE is used to remove a resource.

4. Discovery

To discover the location of the feeds exposed by an Atom Protocol service, the client must locate and request a Service Description Document (Section 6).

```
   Client                      Server
   |                                |
   |  1.) GET Service URI           |
   |------------------------------->|
   |                                |
   |  2.) Service Description Doc   |
   |<-------------------------------|
   |                                |
```

1. The client sends a GET request to the Service Description URI.

2. The server responds with a Service Description Document containing the locations of feeds provided by the service. The content of this document can vary based on aspects of the client request, including, but not limited to, authentication credentials.

5. Listing

Once the client has discovered the location of a feed in the outline, it can request a listing of the feed's entries. However, a feed might contain an extremely large number of entries, so servers are likely to list a small subset of them by default.

```
      Client                      Server
       |                                 |
       |  1.) GET to Atom Feed URI       |
       |-------------------------------->|
       |                                 |
       |  2.) 200 OK, Atom Feed Doc      |
       |<--------------------------------|
       |                                 |
```

1. The client sends a GET request to the Atom Feed's URI.
2. The server responds with an Atom Feed Document containing a full or partial listing of the feed's membership.

6. Authoring

After locating a feed, a client can add entries by sending a POST request to the feed; other changes are accomplished by sending HTTP requests to each entry.

6.1. Create

```
      Client                      Server
       |                                 |
       |  1.) POST Entry to Feed URI     |
       |-------------------------------->|
       |                                 |
       |  2.) 201 Created @ Location     |
       |<--------------------------------|
       |                                 |
```

1. The client sends an Atom Entry to the server via HTTP POST. The Request URI is that of the Atom Feed.
2. The server responds with a response of "201 Created" and a "Location" header containing the URI of the newly-created Atom Entry.

6.2. Read

```
      Client                          Server
      |                                 |
      |  1.) GET or HEAD to Entry URI   |
      |-------------------------------->|
      |                                 |
      |  2.) 200 OK Atom Entry          |
      |<--------------------------------|
      |                                 |
```

1. The client sends a GET (or HEAD) request to the entry's URI.
2. The server responds with an Atom Entry document.

6.3. Update

```
      Client                          Server
      |                                 |
      |  1.) PUT to Entry URI           |
      |-------------------------------->|
      |                                 |
      |  2.) 200 OK                     |
      |<--------------------------------|
      |                                 |
```

1. The client PUTs an updated Atom Entry Document to the entry's URI.
2. The server responds with a successful status code.

6.4. Delete

```
      Client                          Server
      |                                 |
      |  1.) DELETE to Entry URI        |
      |-------------------------------->|
      |                                 |
      |  2.) 204 No Content             |
      |<------------------------------|
      |                               |
```

1. The client sends a DELETE request to the entry's URI.
2. The server responds with successful status code.

6.5. Success and Failure

HTTP defines classes of response. HTTP status codes of the form 2xx signal that a request was successful. HTTP status codes of the form 4xx or 5xx signal that an error has occurred, and the request has failed. Consult the HTTP specification for more detailed definitions of each status code.

7. Atom Protocol Feeds

7.1. GET

Feeds can contain extremely large numbers of resources. A naive client such as a web spider or web browser would be overwhelmed if the response to a GET contained every entry in the feed, and the server would waste large amounts of bandwidth and processing time on clients unable to handle the response. As a result, responses to a simple GET request represent a server-determined subset of the entries in the feed.

An example Atom Protocol feed:

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:pub="http://purl.org/atom/app#">
  <title>My Posts1</title>
  <id>urn:uuid:ce61592c-14e2-4557-978e-dfbd444aefa6</id>
  <updated>2005-12-21T04:11:00-08:00</updated>
  <!-- 0 or more atom:entry elements follow -->
  <entry>
    <title type="text">title 25</title>
    <updated>2005-12-21T04:11:00-08:00</updated>
    <author>
      <name>Foo</name>
    </author>
    <summary>It started out looking simple enough...</summary>
    <id>urn:uuid:941e12b4-6eeb-4753-959d-0cbc51875387</id>
    <pub:edit href="./entry7.atom"/>
    <link href="/permalink7.html" />
   </entry>
   ...
</feed>
```

Each member entry is represented by an atom:entry element, but those entries are not an editable representation of the entry. To retrieve the source representation of the entry, clients send a GET request to the URI found in each entry's pub:edit element (see Section 4.3.1). Derived resources are located by examining an entry's atom:link elements.

7.2. POST

An Atom Protocol feed also accepts POST requests. The client POSTs a representation of the desired resource to the APP feed. Some feeds only accept POST requests with certain media-types, so a POST request could result in a response with a status code of 415 ("Unsupported Media Type"). In the case of a successful creation, the status code is 201 ("Created").

Example request creating a new entry in a feed:

```
       POST /collection HTTP/1.1
       Host: example.org
       User-Agent: Cosimo/1.0
       Content-Type: application/atom+xml
       Content-Length: nnnn

       ...data...
```

Example response.

```
       HTTP/1.1 201 Created
       Date: Mon, 21 Mar 2005 19:20:19 GMT
       Server: CountBasic/2.0
       ETag: "4c083-268-423f1dc6"
       Location: http://example.org/stuff/foo13241234.atom
```

8. Media Feeds

The entries within Media Feeds do not represent uniform types of content. For example, they might contain JPEG images, text documents, MPEG movies, or any other type of resource the server allows.

8.1. GET

Media Feeds return an Atom feed much like the textual Atom Protcol feeds described above, but with a few additions. The entries also contain an atom:content element with a 'src' attribute pointing to the media resource. This URI can be used to edit the uploaded media resource, using PUT and DELETE. Such entries may contain edit links used to edit the entry metadata. As with any Atom entry, related and derived resources can be located by inspecting an entry's atom:link elements.

An example Media Feed:

```
       <feed xmlns="http://www.w3.org/2005/Atom"
             xmlns:pub="http://purl.org/atom/app#">
         <title>My Posts1</title>
         <author>
            <name>Foo</name>
         </author>
         <id>urn:uuid:ce61592c-14e2-4557-978e-dfbd444aefa6</id>
         <updated>2005-12-21T04:11:00-08:00</updated>
         <!-- 0 or more atom:entry elements follow -->
         <entry>
           <title type="text">Title25</title>
           <updated>2005-12-21T04:11:00-08:00</updated>
           <id>urn:uuid:941e12b4-6eeb-4753-959d-0cbc51875387</id>
           <link href="/permalink7.html" type="text/html" />
           <link href="/stuff/public/beach.jpg" type="image/jpg"
                 title="Low res public version" />
           <summary>This was awesome.</summary>
           <content src="http://example.org/beach.tiff" />
         </entry>
         ...
       </feed>
```

The Atom Syndication Format requires that each such entry contain an atom:title and atom:summary element. This requirement can be challenging to meet without requiring users to enter tedious metadata, but servers should attempt to provide textual data about the resource in the interests of accessibility. The atom:title element will likely be provided by the client, as a way for users to associate their local resources with those they have uploaded to the server (see POST below).

8.2. POST

To add an entry to a Media Feed, clients POST the resource to the Media Feed's URI. Clients should provide a 'Title' request header to provide the server with a short string identifying the resource to users. Clients may include a 'Content-Description' header [RFC2045] providing a more complete description of the content. In addition, servers may inspect the POSTed entity for additional metadata to be exposed in an atom:entry element when listed in a Media Feed. For example, the server might inspect a JPEG file for EXIF headers containing creator data.

An example request:

```
      POST /collection HTTP/1.1
      Host: example.org
      User-Agent: Cosimo/1.0
      Content-Type: image/tiff
      Content-Length: nnnn
      Title: A Trip to the beach
      Content-Description: It was so fun.

      ...binary data...
```

An example response:

```
      HTTP/1.1 201 Created
      Date: Mon, 21 Mar 2005 19:20:19 GMT
      Server: CountBasic/2.0
      ETag: "4c083-268-423f1dc6"
      Location: http://example.org/stuff/beach.tiff
```

[@@ deal with response ambiguity noted in WG]

9. Service Description

Many Atom Protocol applications require a basic resource layout in order to ease configuration requirements. Servers use Service Description documents to convey information about related groups of Atom Protocol feeds. On a blogging service, for example, each group might represent a distinct blog and associated resources.

Example Service Description document:

```
   <app xmlns="http://purl.org/atom/app#">
     <service name="My Blog" class="feed"
      href="http://example.com/entries">
      <service name="Photos" class="media feed"
       href="http://example.com/photos"/>
      <service name="Drafts" class="feed"
       href="http://example.com/drafts"/>
     </service>
     <service class="feed" name="Sidebar Blog"
      href="http://example.org/details"/>
      </app>
```

Servers are not required to expose a Service Description document, but experimental deployment experience has shown that a single document which signals some basic information about the server's configuration can greatly simplify client implementations. The simplest useful Service Description document shows the location of a single resource:

```
      <app>
        <service name="My Blog" class="feed"
         href="http://blog.example.com/app.cgi"/>
      </app>
```

If another service is added, the document can be upgraded to reflect new resources.

```
      <app>
        <service name="My Blog" class="feed"
         href="http://blog.example.com/app.cgi"/>
        <service name="Another Blog" class="feed"
         href="http://another.example.com/app.cgi"/>
      </app>
```

Finally, more extensive services could require some amount of hierarchical grouping.

```
      <app>
        <service name="My Blog" class="feed"
         href="http://blog.example.com/app.cgi">
          <service name="Photos" class="media feed"
           href="http://example.com/photos"/>
        </service>
        <service name="Other Things">
          <service name="Another Blog" class="feed"
           href="http://another.example.com/app.cgi"/>
          <service name="A Third Blog" class="feed"
           href="http://third.example.com/app.cgi"/>
        </service>
      </app>
```

This example shows that links to APP feeds can appear in <service> elements used to group other resources. The <service> element named "Other Things" does not contain an 'href' attribute, so it functions as a simple named group of the services it contains.

9.1. Categories

[@@ tbd]

9.2. Document Format

Service Description documents MUST be well-formed XML [W3C.REC-xml-20040204].

The root element of an APP Service Description Document is "<app>". This specification does not define any attributes of the <app> element, but the element can have any number of attributes.

Zero or more <service> elements appear as child elements of <app>. Also, <service> elements may contain zero or more <service> elements. This specification defines three attributes of the <service> element. <service> elements contain at least a 'name' or 'href' attribute. Additional service properties too large or structured to include in attribute values could appear as child elements of the service element.

<app> elements can contain any number of elements that are not <service> elements, and <service> elements can contain any number of elements that are not <service> elements.

9.2.1. The 'name' Attribute

The 'name' attribute contains a short string describing the service element. Entities such as "&amp;" and "&lt;" represent their corresponding characters ("&" and "<" respectively), not markup.

9.2.2. The 'href' Attribute

The 'href' attribute contains an IRI reference interpreted relative to the in-scope base IRI [RFC3987]. Most protocols require URIs [RFC3986], so IRIs usually need to be converted to URIs before being dereferenced.

9.2.3. The 'class' Attribute

The 'class' attribute contains a space-separated list of strings used to classify the service element. This specification defines two values for the 'class' attribute:

- o feed
- o media feed

These values correspond to standard feeds and media feeds, respectively. If the 'class' attribute is not present, the <service> element can be processed as if the attribute were present with a value of 'feed'.

9.2.4. Relax NG Schema

Service Description documents conform to the schema below.

```
default namespace = "http://purl.org/atom/app#"
start = app

app = element app {
  anyAttribute*,
  (service* & anyElement*)
}

service = element service {
  (nameAtt | hrefAtt), anyAttribute*,
  (service* & anyElement*)
}

nameAtt = attribute name { text }
hrefAtt = attribute href { text }
classAtt = attribute class { text }

anyElement = element * { (anyAttribute | text | anyElement)* }
anyAttribute = attribute * { text }
```

9.2.5. Extending Service Description

The Service Description document format can be freely extended by adding attributes and elements not defined by this specification.

Valid Service Description document with extensions:

```
<app xmlns="http://purl.org/atom/app#" foo="bar">
  <blog-userid>42</blog-userid>
  <service name="Baz" qux="hmmm" href="http://example.com">
    <some-other-extension>hmm</some-other-extension>
  </service>
</app>
```

Additional service properties too large or structured to include in attribute values could appear as child elements of the <service> or <app> elements. <app> elements may contain any number of elements that are not <service> elements, and <service> elements may contain any number of elements that are not <service> elements.

9.2.6. User Agent Conformance

Foreign markup is markup not defined by this specification.

Software consuming Service Description documents must not halt processing when any foreign markup is encountered. Software may ignore the markup and process any content of foreign elements as though the surrounding markup were not present. For example, software may process

```
<app>
  <workspace>
    <service name="My Blog"
     href="http://example.com/entries">
      <service name="Photos" class="media feed"
       href="http://example.com/jpgs"/>
      <view title="Archives" seek="...">
      <view title="2005" href="..." />
        ...
      </view>
    </service>
  </workspace>
</app>
```

as though the <workspace> and <view> elements were not present.

Software conforming to this specification may halt processing when documents that do not conform to the schema are encountered.

10. IANA Considerations

[@@ fill out in for application/sd+xml (service description)]

11. Security Considerations

12. Informative References

Appendix A. Acknowledgements

This draft is a variant of the in-progress Atom Publishing Protocol specification from the IETF Atompub WG, and owes a debt to the WG's members.

Appendix B. Change History

```
   -06:  Change service description format.
      Change IPR terms to full3978
   -interlude:  More unproductive WG thrashing.
   -05:  Death to collections!
      Switch APPO instead of XOXO.
      State the obvious about the extension elements.
      Remove RFC2119 reference.
      Change "Normative References" to "References".
   -04:  Add pub:control element.
      Reword collection POST.
      Prophesize about atom:id.
   -03:  Remove search/query capabilities added in -02 Drop round-
      tripping.  Most of them were writable, some folks wanted to edit
      atom:updated, that leaves atom:id, and that seems foolish to try
      and edit, so go ahead and try it if you think you can.
   -02:  Add search/query capabilities.
   -01:  Split from WG draft, cut SOAP, and much other cruft.
   -interlude:  Becomes WG draft.
   -00:  Split from WG draft.
```

Author's Address

Robert Sayre

Email: rfsayre@boswijck.com

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.