Network Working Group   J. Gregorio, Ed.
Internet-Draft   BitWorking, Inc
Expires: September 19, 2005   R. Sayre, Ed.
                                                Boswijck Memex Consulting
                                                           March 18, 2005

The Atom Publishing Protocol
draft-ietf-atompub-protocol-03.txt

Status of this Memo

Copyright Notice

Abstract

This memo presents a protocol for using XML (Extensible Markup Language) and HTTP (HyperText Transport Protocol) to edit content.

The Atom Publishing Protocol is an application-level protocol for publishing and editing Web resources belonging to periodically

updated websites. The protocol at its core is the HTTP transport of Atom-formatted representations. The Atom format is documented in the Atom Syndication Format (draft-ietf-atompub-format-06.txt).

Editorial Note

To provide feedback on this Internet-Draft, join the atom-syntax mailing list (http://www.imc.org/atom-syntax/index.html) [1].

Table of Contents

1. Introduction

The Atom Publishing Protocol is an application-level protocol for publishing and editing Web resources using HTTP [RFC2616] and XML.

1.1 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.2 Terminology

Atom Entry: An Atom Entry is a fragment of a full Atom feed. In this case, the fragment is a single 'entry' element and all its child elements. Each Atom Entry describes a single Web resource, providing metadata and optionally a textual representation of that resource.

2. The Atom Publishing Protocol Model

The Atom Publishing Protocol is an application-level protocol for publishing and editing Web resources. The primary way of interaction in the Atom Publishing Protocol is by managing collection of resources. All collections support the same basic methods of interaction. In addition, the resources belonging to collections also share the same interaction patterns. Using the common HTTP verbs provides a pattern for working with all such Web resources:

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

2.1 Atom Collections

An Atom collection is a set of items all of the same type ("members" of the collection), where the "type" may be, for example: Atom entry, category, template, "simple resource", or any other classification of web resource.

Each collection has a URI which is given in the introspection file. A GET on the collection URI MUST produce a collection document as defined in "3.X.1 Collection Document." That document describes PART OF the state of the collection.

All the members of a collection have an "updated" property, and the collection is considered to be ordered by this property. A single

collection document may not contain all of the members of a collection. If a collection document is the response of a non-partial GET request, and does not contain all of the members of a collection, then it will contain the URI of the next collection document which will contain more of the collection members. By traversing this list of collection documents a client can obtain all of the members of a collection. The 'next' attribute will not be present in the response to a partial GET request.

2.1.1 Usage

Below two usages are outlined for Atom Collections. They are here to highlight common idioms for interacting with a Collection Resource and not a normative interaction pattern.

The Atom Collection can be used by clients in two ways. In the first case the client has attached to a site for the first time and is doing an initial syncronization, that is, retrieving a list of all the members of the collections and possibly retrieving all the members of the collection also. The client can perform a non-partial GET on the collection resource and it will receive a collection document that either contains all the member of the collection, or the collection document root element 'collection' will contain a 'next' attribute pointing to the next collection document. By repeatedly following the 'next' attribute from document to document the client can find all the members of the collection.

In the second case the client has already done an initial sync, and now needs to re-sync, because the client was just restarted, or some time has passed since a re-sync, etc. The client does a partial GET on the collection document, supplying a Range header that begins from the last time the client sync'd to the current time. The collection document returned will contain only those members of the collection that have changed since the last time the client syncronized.

2.1.2 Client and Server Interaction

[[anchor5: ...]]

This document does not specify the form of the URIs that are used. The URI space of each server is controlled, as defined by HTTP, by the server alone. What this document does specify are the formats of the files that are exchanged and the actions that can be performed on the URIs embedded in those files.

3. Functional Specification

3.1 Collections

3.1.1 Collection Document

A collection document is rooted by a <collection> element. A collection element may have any number of <member> elements as children; each such element identifies a member of the collection. In some situations, a collection document may not contain every member of the collection itself.

Whether complete or partial, the members in a collection document MUST constitute a consecutive sequence of the collection's members, ordered by their "updated" properties. That is, a collection document MUST contain a contiguous subset of the members of the collection ordered by their 'updated' property.

3.1.2 Elements in a Collection Document

A collection document MAY contain zero or more 'member' elements.

Each 'member' element MUST include an 'href' attribute identifying a URL of the member resource. The 'href' URI of a member resource is an "EditURI" under the terms of section 2, and MUST respond to the same HTTP methods as such an EditURI.

Each 'member' element MAY include an "hrefreadonly" attribute. This optional attribute identifies a URI which, on a GET request, responds equivalently to how the "href" URI would respond to the same request. Clients SHOULD NOT apply to this URI any HTTP methods that would be expected to modify the state of the resource (e.g. PUT, POST or DELETE). A PUT or POST request to this URI MAY NOT affect the underlying resource. If the "hrefreadonly" attribute is not given, its value defaults to the "href" value. If the "hrefreadonly" attribute is present, and its value is an empty string, then there is no URI that can be treated in the way such a value would be treated.

Clients SHOULD use the "href" value to manipulate the resource within the context of the APP itself. Clients SHOULD prefer the "hrefreadonly" value in any other context. For example, if the resource is an image, a client may replace the image data using a PUT on the "href" value, and may even display a preview of the image by fetching the "href" URI. But when creating a public, read-only reference to the same image resource, the client should use the "hrefreadonly" value. If the "hrefreadonly" value is an empty string, the client SHOULD NOT make public reference to the "href" value.

Each 'member' element MUST include a 'title' attribute, whose value

is a human-readable name or description for the item. The values of 'title' attributes are not required to be unique across all members of a collection.

Each 'member' element MUST include an 'updated' attribute, whose value is the 'updated' property of the collection member whose format MUST conform to the date-time BNF rule in [RFC3339].

3.1.3 Collection Requests

3.1.3.1 Range: Header

HTTP/1.1 allows a client to request that only part (a range of) the collection to be included within the response. HTTP/1.1 uses range units in the Range header field. A collection can be broken down into subranges according to the members 'updated' property. If a Range: header is present in the request, its value explictly identifies the a time interval interval in which all the members 'updated' property must fall to be included in the response.

```
Range = "Range" ":" ranges-specifier
```

The value of the Range: header should be a pair of ISO 8601 dates, separated by a slash character; either date may be optionally omitted, in which case the range is understood as stretching to infinity on that end.

```
ranges-specifier = updated-ranges-specifier
updated-ranges-specifier = updated-unit "=" updated-range
updated-unit = "updated"
updated-range = [iso-date] "/" [iso-date]
```

The response to a collection request MUST be a collection document, all of whose 'member' elements fall within the requested range. If no members fall in the requested range, the server MUST respond with a collection document containing no 'member' elements.

3.1.3.2 Accept-Ranges: Header

The response to a non-partial GET request MUST include an Accept-Ranges header that indicates that the server accepts 'updated' range requests.

```
Accept-Ranges     = "Accept-Ranges" ":" acceptable-ranges
acceptable-ranges = updated-unit ( 1#range-unit )
```

3.2 Introspection

There are many different kinds of resources that can be managed through the APP, for example, entries, templates, users, etc. The Service Document is a single document that lists all the facets of the APP that a site supports and also contains the URIs of all those resources.

3.2.1 Service Document

The Service Document lists the resources that each site makes available. The Service Resource returns an Service Document in response to a GET request. Here is an example of an Service Document.

```
<?xml version="1.0" encoding='utf-8'?>
<service version="0.3" xmlns="http://purl.org/atom/ns#">
  <workspace title="Main Site" >
    <collection rel="entries" name="Entries"
      href="http://example.org/reilly/feed" />
    <collection rel="categories" name="Categories"
      href="http://example.org/reilly/cat" />
    <collection rel="templates" name="Templates"
      href="http://example.org/reilly/tmpl" />
    <collection rel="users" name="Users"
      href="http://example.org/reilly/users" />
    <collection rel="resource" name="Pictures"
      href="http://example.org/reilly/pic" />
  </workspace>
  <workspace title="b-links">
    <collection rel="entries" name="Entries"
      href="http://example.org/reilly/feed" />
    <collection rel="http://example.net/booklist" name="Books"
      href="http://example.org/reilly/books" />
  </workspace>
</service>
```

- entries
- resource
- categories
- templates
- users

The default for the rel attribute is 'resource'. Extensibility for 'rel' values is handled in the same manner as PaceFieldingLinks. Each 'collection' element in 'workspace' represents a single facet of the APP. While a site must fully support each facet they list in their Service Document, a site does not need to support all the facets in this RFC. Additionally, new facets may be added either

through vendor extension or follow-on RFCs.

3.2.1.1 Service Documet Elements

The "service" element is the document element of a Service Document, acting as a container for service data associated with possibly multiple workspaces. Its only child elements MUST be one or more 'workspace' elements. The 'service' element MUST have a single attribute 'version' whose content indicates the version of the Atom specification that the document conforms to. The content of this attribute is unstructured text. The version identifier for this specification is "1.0".

The 'workspace' element element contains information elements about the collections of resources available for editing. The only children of 'workspace' MUST be one or more "collection" elements. The 'workspace' element MUST have a single attribute 'title' whose content MUST NOT be empty and which is a human-readable name for the workspace.

The 'collection' element describes various typed groups of resources available for editing or adding to.

3.3 Entry Collection

Entries are managed through collections and as such entry collection and entries that are members of a collection must support all the operations enumerated above.

An Edit Resource is used to edit a single entry. Each entry that is editable MUST have a unique URI. This URI supports both GET and PUT and they are used in tandem for an editing cycle. The client GETs the representation which is formatted as an Atom entry. The client may then update the entry and then PUT it back to the same URI. The PUT will cause all the related resources to be updated, for example, the HTML representation.

Note that the value of the content element in the Atom entry does not have to exactly match the content element for the same entry when it is represented in an Atom feed. For example, a server may allow the client to post entries whose content is formatted as WikiML, yet the server may clean up such markup and transform it into well-formed XHTML before placing it in the publicly available Atom feed. Another scenario is summaries--the EditURI is for editing the full content of an entry, but the server may only present excerpts when it produces an Atom feed.

A client will send a DELETE to the EditURI to delete an entry.

3.3.1 Locating

For editing a site Entry, the link tag is used. Note that a link tag is used in both HTML and in the Atom format. A link tag of the following format points to the EditURI for a site. In HTML, the link tags for editing are always found in the head element, while in Atom they may appear as children of the entry elements.

```
<link rel="service.edit"
type="application/atom+xml"
href="URI for Editing goes here"
title="Readable desc of the entry." />
```

Note: The critical characteristic of this link tag is the @rel of 'service.edit' and the @type of 'application/atom+xml'.

3.4 Simple Resource Collection

Simple Resources are managed through collections and as such simple reource collections and simple resources that are members of the collection must support all the operations enumerated above. Simple Resources can be images, templates, and any other non-entry resources.

3.4.1 Locating

For creating a new non-entry resource, the link tag is used. Note that a link tag is used in both HTML and in the Atom format. A link tag of the following format points to the ResourcePostURI for a site. In HTML the link tags are always found in the head element, while in Atom they may appear as children of the Feed and entry elements.

```
<link rel="resource.post" href="URI for Resource Posting goes here"
title="The name of the site.">
```

3.4.2 Request

The request contains a resource, sent through a standard HTTP POST, e.g.:

```
POST /_do/exampleblog/post_resource HTTP/1.1
Host: www.example.com
Content-Type: image/jpeg
Content-Length: nnn

...raw bytes of image go here...
```

3.5 Atom Request and Response Body Constraints

The Atom format is used as the representation of all the resources in this specification. As it is used in differing contexts, there are different constraints of which elements may be present, and how their values should be interpreted.

3.5.1 id

PostURI MUST NOT be present.
FeedURI MUST be present.
EditURI
   GET MUST be present.
   PUT MUST be present.

3.5.2 link

PostURI MAY be present. Servers MAY use the information to determine the URI of the created resource. Relative URLs are to be interpreted relative to xml:base.
FeedURI MUST be present.
EditURI
   GET MUST be present.
   PUT MUST be present.

3.5.3 title

PostURI MUST be present. The element may be empty, to explicitly indicate "no title". Servers SHOULD NOT try to generate a title if one is not provided. The type attribute MAY be present, and if not it defaults to "text/plain". If present, it MUST represent a MIME type that the server supports. The mode attribute MAY be present. If not present, it defaults to "xml". If present, it MUST be "xml", "base64", or "escaped".
FeedURI MUST be present.
EditURI
   GET MUST be present.
   PUT MUST be present. The element may be empty, to explicitly indicate "no title". Servers SHOULD NOT try to generate a title if one is not provided.

3.5.4 summary

PostURI MAY be present. If not present, the server is welcome to produce its own summary. If present but empty, the server SHOULD NOT generate a summary of its own. The type attribute MAY be present. If not, it defaults to "text/plain". If present, it must represent a MIME type that the server supports. The mode

      attribute MAY be present and defaults to "xml". If present, it
      must be "xml","base64", or "escaped".
   FeedURI MAY be present.
   EditURI
      GET MAY be present.
      PUT MAY be present. The element may be empty, to explicitly
         indicate "no summary". Servers SHOULD NOT try to generate a
         title if one is not provided.

3.5.5 content

   PostURI MAY be present but may be empty, to explicitly indicate "no
      content". The type attribute MAY be present, but defaults to
      "text/plain" if not present. It must represent a MIME type that
      the server supports. The MODE attribute may be present and
      defaults to "xml" if not present. It must be "xml","base64", or
      "escaped".
   FeedURI MAY be present.
   EditURI
      GET MAY be present.
      PUT MAY be present. The element may be empty, to explicitly
         indicate "no content".

3.5.6 issued

   PostURI MUST be present, but may be empty, in which case it signifies
      "now" in the time zone of the server.
   FeedURI MUST be present.
   EditURI
      GET MUST be present.
      PUT MUST be present. Server policy determines if an updated time
         is accepted.

3.5.7 modified

   PostURI MUST NOT be present.
   FeedURI MAY be present.
   EditURI
      GET MAY be present.
      PUT MAY be present. The element may be empty, to explicitly
         indicate that 'now' on the server time is to be used.

3.5.8 created

   PostURI MAY be present.

```
FeedURI MAY be present.
EditURI
   GET MAY be present.
   PUT MAY be present.  The server may or may not accept an updated
      value.  If the server does not allow updating the issued time
      then any PUT request with a different issued value MUST be
      rejected.
```

3.5.9 author

```
PostURI MAY be present.  If not present, the server determines the
   author.  If present, and conflicting with valid values as
   determined by the server, then the server may change the value of
   author.
FeedURI MAY be present.
EditURI
   GET MAY be present.
   PUT MAY be present.
```

3.5.10 contributor

```
PostURI MAY be present.
FeedURI MAY be present.
EditURI
   GET MAY be present.
   PUT MAY be present.
```

3.5.11 generator

```
PostURI MUST be present and contain a URI.  The value of the element
   indicates the code base used to create this request.  MUST also
   have an attribute 'version' with a version number.
FeedURI MUST NOT be present.
EditURI
   GET MUST NOT be present.
   PUT MUST NOT be present.
```

3.6 Securing the Atom Protocol

All instances of publishing Atom entries SHOULD be protected by authentication to prevent posting or editing by unknown sources. Atom servers and clients MUST support one of the following authentication mechanisms, and SHOULD support both.

o HTTP Digest Authentication [RFC2617]
o [@@TBD@@ CGI Authentication ref]

Atom servers and clients MAY support encryption of the Atom session

using TLS [RFC2246].

There are cases where an authentication mechanism may not be required, such as a publicly editable Wiki, or when using the PostURI to post comments to a site that does not require authentication to create comments.

3.6.1 [@@TBD@@ CGI Authentication]

This authentication method is included as part of the protocol to allow Atom servers and clients that cannot use HTTP Digest Authentication but where the user can both insert its own HTTP headers and create a CGI program to authenticate entries to the server. This scenario is common in environments where the user cannot control what services the server employs, but the user can write their own HTTP services.

4. Security Considerations

Because Atom is a publishing protocol, it is important that only authorized users can create and edit entries.

The security of Atom is based on HTTP Digest Authentication and/or [@@TBD@@ CGI Authentication]. Any weaknesses in either of these authentication schemes will obviously affect the security of the Atom Publishing Protocol.

Both HTTP Digest Authentication and [@@TBD@@ CGI Authentication] are susceptible to dictionary-based attacks on the shared secret. If the shared secret is a password (instead of a random string with sufficient entropy), an attacker can determine the secret by exhaustively comparing the authenticating string with hashed results of the public string and dictionary entries.

See RFC 2617 for more detailed description of the security properties of HTTP Digest Authentication.

@@TBD@@ Talk here about using HTTP basic and digest authentication.

@@TBD@@ Talk here about denial of service attacks using large XML files, or the billion laughs DTD attack.

5. IANA Considerations

This document has no actions for IANA.

6. Appendix A - SOAP Enabling

All servers SHOULD support the following alternate interface mechanisms to enable a wider variety of clients to interact with Atom Publishing Protocol servers. The following requirements are in addition to the ones listed in the Functional Specification Section. If a server supports SOAP Enabling then it MUST support all of the following.

6.1 Servers

1. All servers MUST support the limited use of the SOAPAction HTTP Header as described below in the Client section.
2. All servers MUST be able to process well formed XML. Servers need not be able to handle processing instructions or DTDs.
3. Servers MUST accept content in a SOAP Envelope, and if they receive a request that is wrapped in a SOAP Envelope then they MUST wrap their responses in SOAP envelopes or produce a SOAP Fault.

6.2 Clients

1. Clients SHOULD use the appropriate HTTP Method when possible. When not possible, they should use POST and include a SOAPAction HTTP header which is constrained as follows:
2. SOAPAction: "http://schemas.xmlsoap.org/wsdl/http/[METHOD]"
3. Where [METHOD] is replaced by the desired HTTP Method.
4. Clients MAY wrap their XML payload in a SOAP Envelope. If so, they must also wrap it in an element which exactly matches the HTTP Method.

7. Appendix B - Examples

7.1 Example for a weblog

Fill this in with an example for how all the above is used for a weblog. Start with main HTML page, link tag of type service.feed to the 'introspection' file. 1. Creating a new entry 2. Finding an old entry 3. editing an old entry 4. commenting on a entry (via HTML and Atom)

7.2 Example for a wiki

Fill this in like above but for a wiki.

8. Revision History

draft-ietf-atompub-protocol-03 - Incorporates PaceSliceAndDice3 and

PaceIntrospection.

draft-ietf-atompub-protocol-02 - Incorporates Pace409Response, PacePostLocationMust, and PaceSimpleResourcePosting.

draft-ietf-atompub-protocol-01 - Added in sections on Responses for the EditURI. Allow 2xx for response to EditURI PUTs. Elided all mentions of WSSE. Started adding in some normative references. Added the section "Securing the Atom Protocol". Clarified that it is possible that the PostURI and FeedURI could be the same URI. Cleaned up descriptions for Response codes 400 and 500.

Rev draft-ietf-atompub-protocol-00 - 5Jul2004 - Renamed the file and re-titled the document to conform to IETF submission guidelines. Changed MIME type to match the one selected for the Atom format. Numerous typographical fixes. We used to have two 'Introduction' sections. One of them was moved into the Abstract the other absorbed the Scope section. IPR and copyright notifications were added.

Rev 09 - 10Dec2003 - Added the section on SOAP enabled clients and servers.

Rev 08 - 01Dec2003 - Refactored the specification, merging the Introspection file into the feed format. Also dropped the distinction between the type of URI used to create new entries and the kind used to create comments. Dropped user preferences.

Rev 07 - 06Aug2003 - Removed the use of the RSD file for auto-discovery. Changed copyright until a final standards body is chosen. Changed query parameters for the search facet to all begin with atom- to avoid name collisions. Updated all the Entries to follow the 0.2 version. Changed the format of the search results and template file to a pure element based syntax.

Rev 06 - 24Jul2003 - Moved to PUT for updating Entries. Changed all the mime-types to application/x.atom+xml. Added template editing. Changed 'edit-entry' to 'create-entry' in the Introspection file to more accurately reflect it's purpose.

Rev 05 - 17Jul2003 - Renamed everything Echo into Atom. Added version numbers in the Revision history. Changed all the mime-types to application/atom+xml.

Rev 04 - 15Jul2003 - Updated the RSD version used from 0.7 to 1.0. Change the method of deleting an Entry from POSTing <delete/> to using the HTTP DELETE verb. Also changed the query interface to GET instead of POST. Moved Introspection Discovery to be up under Introspection. Introduced the term 'facet' for the services listed

in the Introspection file.

Rev 03 - 10Jul2003 - Added a link to the Wiki near the front of the document. Added a section on finding an Entry. Retrieving an Entry now broken out into it's own section. Changed the HTTP status code for a successful editing of an Entry to 205.

Rev 02 - 7Jul2003 - Entries are no longer returned from POSTs, instead they are retrieved via GET. Cleaned up figure titles, as they are rendered poorly in HTML. All content-types have been changed to application/atom+xml.

Rev 01 - 5Jul2003 - Renamed from EchoAPI.html to follow the more commonly used format: draft-gregorio-NN.html. Renamed all references to URL to URI. Broke out introspection into it's own section. Added the Revision History section. Added more to the warning that the example URIs are not normative.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[1] <http://www.imc.org/atom-syntax/index.html>

Authors' Addresses

Joe Gregorio (editor)
BitWorking, Inc
1002 Heathwood Dairy Rd.
Apex, NC 27502
US

Phone: +1 919 272 3764
Email: joe@bitworking.com
URI: http://bitworking.com/

Robert Sayre (editor)
Boswijck Memex Consulting
148 N 9th St. 4R
Brooklyn, NY 11211
US

Email: rfsayre@boswijck.com
URI: http://boswijck.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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