Network Working Group
Internet-Draft
Expires: December 30, 2004

J. Gregorio, Ed.
BitWorking, Inc
R. Sayre, Ed.
Boswijck Memex Consulting
July 1, 2004

The Atom Publishing Protocol
draft-ietf-atompub-protocol-01.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-00.txt).

Editorial Note

To provide feedback on this Internet-Draft, join the <http://www.imc.org/atom-syntax/index.html>.

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.

PostURI: A URI that is used to create new resources. POSTing an Atom Entry to this URI will create a new resource.

EditURI: A URI that is used to edit a resource. The editing is done using the HTTP verbs GET, PUT and DELETE. The representation of the resource is always that of an Atom Entry.

FeedURI: The URI which identifies an Atom Feed.

2. The Atom Publishing Protocol Model

The Atom Publishing Protocol is an application-level protocol for publishing and editing Web resources. Using the common HTTP verbs provides a pattern for working with all such Web resources:

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

There are three major classes of URIs in this specification: PostURI, FeedURI and EditURI. This specification defines the expected actions for each of the methods listed. A URI MAY support methods not listed here. For example, an EditURI could support a POST or OPTIONS

method. However, what those methods do is beyond the scope of this specification.

- EditURI: PUT, GET, DELETE
- PostURI: POST
- FeedURI: GET

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 PostURI

The PostURI is used to create entries. These can be either full entries, such as a weblog post, or they can be comments, or even a wiki page. The client POSTs a filled-in Atom Entry to this URI. If the request is successful, one or more Web resources MAY be created. For example, POSTing an Atom entry to a PostURI may create two new Web resources, an HTML representation and an Atom representation.

3.1.1 Locating the PostURI

The PostURI can be discovered in a link element with an @rel of 'service.post'. The link element containing a PostURI used to create a new entry MAY be discovered in three different places. The first place it may be found is in a <link> element in the 'head' element of an HTML document.

The second place a PostURI may be found an atom:link element that is a child of the atom:feed element. The third place a PostURI may be found is in the atom:link element of an atom:entry.

@@ TBD @@ - Discuss subordinate resources and what a PostURI means based on where the URI was found.

```
<link rel="service.post"
      type="application/atom+xml"
      href="URI for Posting goes here"
      title="The name of the site.">
```

3.1.2 Request

The request contains a filled-in Atom entry, subject to the constraints in section Section 3.5.

3.1.3 Response

The possible status codes from a POST are 201, 303, 400, 404, 410 and 500.

3.1.3.1 Response code 201

Response includes a Location: header with the URI of the created

resource, i.e. the URI used to edit the entry, as opposed to the URI used to display the content. The body of the response will contain the entry "filled-in" with time stamps and any other data the server chooses to reveal. This must contain enough information to enable a client to issue a subsequent PUT to this location. Note that the server may chose to omit the content in the response, particularly if it is large.

3.1.3.2 Response code 303

The body of this response does not contain the filled-in Entry, but the filled-in Entry can be found under a different URI and can be retrieved using a GET method on that resource. The URI SHOULD be given by the Location field in the response.

3.1.3.3 Response code 400

Indicates that the server believes that the data sent constitutes an invalid request. As an example, the data posted may not be well-formed XML. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

3.1.3.4 Response code 500

Indicates that the server detected an internal error on the server processing this request (such as an unhandled exception). The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

3.2 EditURI

An EditURI 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.

3.2.1 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'.

3.2.2 Request

A PUT request, and a GET response both contain a filled-in Atom entry, subject to the constraints in section Section 3.5.

The expected status codes from a GET are 200, 301, 307, and 500. 400, 404, and 410 are also possible.

The expected status codes from a PUT are 2xx, 301, 307, 500 and 501. 400, 404, and 410 are also possible.

3.2.2.1 Successful Requests

Servers MUST indicate successful GET requests with a 200 response.

Servers MUST indicate successful PUT requests with a 2xx response. Servers MAY include additional information in the PUT response.

Clients SHOULD NOT expect any additional information in a PUT response.

3.2.2.2 Response code 301

The entry has moved permanently, the new URI is given in the Location header. The client SHOULD retry the GET using the URI returned in the Location header. When a PUT operation is attempted the user agent should prompt the user before attempting the PUT on the URI returned in the Location header.

3.2.2.3 Response code 307

The entry has moved temporarily, the new URI is given in the Location header. The client SHOULD retry the GET using the URI returned in the Location header. When a PUT operation is attempted the user agent should prompt the user before attempting the PUT on the URI returned in the Location header.

3.2.2.4 Response code 401

Indicates that the server believes that the data sent constitutes an invalid request. As an example, the data posted may not be well-formed XML. The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

3.2.2.5 Response code 410

Indicates that the requested resource is gone permanently. The client SHOULD NOT repeat the request again.

3.2.2.6 Response code 500

Indicates that the server detected an internal error on the server processing this request (such as an unhandled exception). The server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

3.3 FeedURI

The FeedURI is used to retrieve a representation in Atom format. Note that this feed is different from a typical Atom feed in that it contains "link" elements for navigating and manipulating the content of the site. For example there should be a "link" element with rel="next" whose URI points to the next block of entries on the site. Similarly, the feed element can contain a "link" element with rel="service.post", the URI of which is a PostURI. Individual entries should contain "link" elements with rel="service.edit" whose URIs are EditURIs.

This document only uses some of the methods available for each type of URI. For example, the only method described by this document for the FeedURI is GET. Any other method may be supported by the URI types described, but defining their behavior is beyond the scope of this document. In this light you may notice that the PostURI only supports the POST method. It is possible, and allowable, that for some implementations the PostURI and the FeedURI are the same URI.

@@ Editor's Note: @@ Note that the "service.feed" takes the place of the Introspection File and the Search facet in previous versions of the specification. That is, facet discovery, which was previously done by inspecting the Introspection file is now done by looking for "link" tags with an attribute "rel" set to "service.[something]" in the "service.feed" file. At the same time the same representation replaces the search facet by having "link" tags that point to other feeds using well-known 'rel' attribute values such as 'next' and 'prev', or the search can branch in multiple directions by specifying multiple link tags with rel="service.feed" and having differing title attributes that announce the kind of search results in that feed.

3.3.1 Locating

A link tag of the following format points to the FeedURI.

```
<link rel="service.feed"
      type="application/atom+xml"
      href="URI goes here"
      title="The name of the site.">
```

3.3.2 Request

The request is a simple GET. No other verbs are currently specified for this URI.

3.3.3 Response

The expected status codes from a GET are 200, 301, 307, and 500. 401, 404, and 410 are also possible.

3.3.3.1 Response code 301

The Feed has moved permanently, the new URI is given in the Location header. The client SHOULD do a GET on the URI returned in the Location header.

3.3.3.2 Response code 307

The Feed has moved temporarily, the new URI is given in the Location header. The client SHOULD do a GET on the URI returned in the Location header.

3.4 Link Tag

The link tag is used in both HTML and Atom formats. There are slight differences between the two usages. Here are the commonalities, differences, and a list of well-known values for the rel attribute.

<http://www.w3.org/TR/html4/struct/links.html#edef-LINK> appears in the 'head' of the document. The 'head' section only allows a linear list of 'link' tags. The Atom format allows 'link' tags as children of both the 'feed' element and of the 'entry' element. Note that this gives the information present in the link tag more context. For example ... @@ TBD @@

3.4.1 rel

This attribute describes the relationship from the current document, be it HTML or Atom, to the anchor specified by the href attribute. The value of this attribute is a space-separated list of link types. Note that these values are case insensitive. When used in concert with type="application/atom+xml", the relations may be interpreted as follows.

alternate: The URI in the href attribute points to an alternate representation of the containing resource.

start: The Atom feed at the URI supplied in the href attribute contains the first feed in a linear sequence of entries.

next: The Atom feed at the URI supplied in the href attribute contains the next N entries in a linear sequence of entries.

prev: The Atom feed at the URI supplied in the href attribute contains the previous N entries in a linear sequence of entries.

service.edit: The URI given in the href attribute is used to edit a representation of the referred resource.

service.post: The URI in the href attribute is used to create new resources.

service.feed: The URI given in the href attribute is a starting point for navigating content and services.

3.4.2 href

URI of the resource being described by this link element.

3.4.3 title

Offers advisory information about the link. Rendered to the user to help them choose among a set of links with the same rel and type attributes.

3.4.4 type

The content type of the resource available at the URI given in the href attribute of the link element. Most of the link types in this specification are on type 'application/atom+xml'.

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.

- HTTP Digest Authentication [RFC2617]
- [@@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-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.

[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.

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.