Network Working Group
Internet-Draft
Expires: November 10, 2005

J. Gregorio, Ed.
BitWorking, Inc
R. Sayre, Ed.
May 9, 2005

The Atom Publishing Protocol
draft-ietf-atompub-protocol-04.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 10, 2005.

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-protocol mailing list (http://www.imc.org/atom-protocol/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 1.0 [W3C.REC-xml-20040204].

2. 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].

3. Terminology

URI/IRI - A Uniform Resource Identifier and Internationalized Resource Identifier, respectively. These terms (and the distinction between them) are defined in [RFC3986] and [RFC3987].

Resource - an item identified by a URI [W3C.REC-webarch-20041215].

Collection Resource - A resource that contains a listing of Member Resources and meets the requirements in Section 5 of this specification.

Member Resource - A resource whose URI is listed by a Collection Resource.

4. The Atom Publishing Protocol Model

The Atom Publishing Protocol operates on collections of Web resources. All collections support the same basic interactions, as do the resources within the collections. The patterns of interaction are based on the common HTTP verbs.

- 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.

4.1 Collections

The APP groups resources into "Collections", which are analogous to the "folders" or "directories" found in many file systems.

4.2 Discovery

To discover the location of the collections exposed by an APP service, the client must locate and request an Introspection Document (Section 8).

```
Client                      Server
|                                |
|  1.) GET Introspection         |
|------------------------------->|
|                                |
|  2.) Introspection Doc         |
|<-------------------------------|
|                                |
```

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

2. The server responds with an Introspection Document containing the locations of collections 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.

4.3 Listing

Once the client has discovered the location of a collection, it can request a listing of the collection's membership. However, collections might be extremely large, so servers are likely to list a small subset of the collection by default.

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

1. The client sends a GET request to the Collection's URI.

2. The server responds with an Atom Feed Document containing a full or partial listing of the collection's membership.

4.4 Authoring

After locating a collection, a client can add entries by sending a request to the collection; other changes are accomplished by sending HTTP requests to its member resources.

4.4.1 Create

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

1. The client sends a representation of a member to the server via HTTP POST. The Request URI is that of the Collection.

2. The server responds with a response of "201 Created" and a "Location" header containing the URI of the newly-created resource.

4.4.2 Read

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

1. The client sends a GET (or HEAD) request to the member's URI.

2. The server responds with an appropriate representation.

4.4.3 Update

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

1. The client PUTs an updated representation to the member's URI.

2. The server responds with a representation of the member's new state.

4.4.4 Delete

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

1. The client sends a DELETE request to the member's URI.

2. The server responds with successful status code.

4.5 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.

5. Collections

An Atom Collection is a set of related resources. All members of a collection have an "updated" property, and the collection is considered to be ordered by this property.

5.1 Collection Documents

An example Collection Document.

```
<?xml version="1.0" encoding='utf-8'?>
<collection xmlns="http://purl.org/atom/app#">
  <member href="http://example.org/1"
          hrefreadonly="http://example.com/1/bar"
          title="Sample 1"
          updated="2003-12-13T18:30:02Z" />
  <member href="http://example.org/2"
          hrefreadonly="http://example.com/2/bar"
          title="Sample 2"
          updated="2003-12-13T18:30:02Z" />
  <member href="http://example.org/3"
          hrefreadonly="http://example.com/3/bar"
          title="Sample 3"
          updated="2003-12-13T18:30:02Z" />
  <member href="http://example.org/4"
          title="Sample 4"
          updated="2003-12-13T18:30:02Z" />
</collection>
```

Atom Collection Documents have the media-type 'application/atomcoll+xml', see Section 11.

5.1.1 Element Definitions

5.1.1.1 The 'app:collection' Element

The 'app:collection' element represents an Atom Collection. A collection document does not necessarily list every member of the collection.

```
appCollection       element app:collection {
      attribute next { text } ?,
      appMember*
   }
```

- o 'app:collection' elements MAY contain any number of 'app:member' elements.

- o 'app:collection' elements MAY contain a 'next' attribute which identifies a collection document containing member elements updated earlier in time.

The members listed 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.

5.1.1.2 The 'app:member' Element

The 'app:member' represents a single member resource.

```
appMember       element app:member {
      attribute title { text },
      attribute href { text },
      attribute hrefreadonly { text } ?,
      attribute updated { text }
   }
```

- o 'app:member' elements MUST include an 'href' attribute, whose value conveys the URI used to edit the member source

- o 'app:member' elements MAY include an "hrefreadonly (Section 5.1.1.3)" attribute.

- o 'app:member' elements MUST include a 'title' attribute, whose value is a human-readable name or description for the item.

- o 'app:member' elements MUST include an 'updated' attribute, whose value is the 'updated' property of the collection member. Its format MUST conform to the date-time production in [RFC3339].

5.1.1.3 The '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.

[[anchor10: Define extensibility for Collection Documents.]]

5.2 Collection Resource

This specification defines two HTTP methods for use with collection resources: GET and POST.

5.2.1 GET

Collections 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 reflected the full membership of the collection, 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 collection's membership.

In addition, the client MAY send a 'Range' header with a range type of 'udpated', indicating the subset of the collection to be returned. The 'Range' header is described in Section 5.2.4.

This specification defines two serializations for Atom Collections. Servers MUST provide both, but MAY also provide additional serializations.

1. Atom Collection Documents (application/atomcoll+xml), Section 5.1.

2. Atom Collection Documents wrapped by a SOAP envelope (application/soap+xml), .

Clients use the HTTP 'Accept' request header to indicate their

preference.

Example Request, with Accept header

```
GET /collection HTTP/1.1
Host: example.org
User-Agent: Agent/1.0
Accept: application/atomcoll+xml
```

Here, the server could return any subset of the collection as an Atom Collection Document.

Example Response, Atom Collection Document

```
HTTP/1.1 200 OK
Date: Fri, 25 Mar 2005 17:15:33 GMT
Last-Modified: Mon, 04 Oct 2004 18:31:45 GMT
ETag: "2b3f6-a4-5b572640"
Accept-Ranges: updated
Content-Length: nnnn
Content-Type: application/atomcoll+xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8"?>
<collection xmlns="http://purl.org/atom/app#">
...
  <member href="http://example.org/1"
          hrefreadonly="http://example.com/1/bar"
          title="Example 1"
          updated="2003-12-13T18:30:02Z" />
...
</collection>
```

Example Request, with SOAP Accept header

```
GET /collection HTTP/1.1
Host: example.org
User-Agent: Cosimo/1.0
Accept: application/soap+xml
```

Here, the server could return any subset of the collection as an Atom Feed Document wrapped by a SOAP envelope.

Example Response, Atom Feed Document wrapped by a SOAP envelope

```
HTTP/1.1 200 OK
Date: Fri, 25 Mar 2005 17:15:33 GMT
Last-Modified: Mon, 04 Oct 2004 18:31:45 GMT
ETag: "2b3f6-a4-5b572640-89"
Accept-Ranges: bytes
Content-Length: nnnn
Content-Type: application/soap+xml; charset="utf-8"

<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Header />
   <env:Body>
      <collection xmlns="http://purl.org/atom/app#">
      ...
      <member href="http://example.org/1"
              hrefreadonly="http://example.com/1/bar"
              title="Example 1"
              updated="2003-12-13T18:30:02Z" />
      ...
      </collection>
   </env:Body>
</env:Envelope>
```

5.2.2 POST

In addition to GET, a Collection Resource also accepts POST requests. The client POSTs a representation of the desired resource to the Collection Resource. Note that some collections only allow members of a specific media-type and a POST MAY generate a response with a status code of 415 ("Unsupported Media Type").

In the case of a successful creation, the status code MUST be 201 ("Created").

Example Request, Create a resource in a collection.

```
POST /collection HTTP/1.1
Host: example.org
User-Agent: Cosimo/1.0
Accept: application/atomcoll+xml
Content-Type: image/png
Content-Length: nnnn
Name: trip-to-beach.png

...binary data...
```

Here, the client is adding a new image resource to a collection. The Name: header indicates the client's desired name for the resource, see Section 5.2.6.

Example Response, resource created successfully.

```
HTTP/1.1 201 Created
Date: Fri, 25 Mar 2005 17:17:11 GMT
Content-Length: nnnn
Content-Type: application/atomcoll+xml; charset="utf-8"
Location: http://example.org/images/trip-to-the-beach-01.png

<?xml version="1.0" encoding="UTF-8"?>
<collection xmlns="http://purl.org/atom/app#">
    <member href="http://example.org/images/trip-to-beach.png"
        hrefreadonly="http://example.com/ed/im/trip-01.png"
        title="trip-to-beach.png"
        updated="2005-03-25T17:17:09Z" />
</collection>
```

5.2.3 Usage Scenarios

These scenarios illustrate common idioms for interactin with Collections.

The Atom Collection can be used by clients in two ways. In the first case the client encounters a Collection 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 members 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.

5.2.4 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. The request range is considered a closed set, that is, if a 'member' element matches one end of the range exactly it MUST be included in the response. If no members fall in the requested range, the server MUST respond with a collection document containing no 'member' elements.

The inclusion of the Range: header in a request changes the request to a "partial GET" [RFC2616].

5.2.5 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 )
```

5.2.6 Name: Header

[[anchor13: this is new...]]

The POST to a Collection Resource MAY contain a Name: header that indicates the clients suggested name for the resource. The server MAY ignore the Name: header or modify the requested name to suit local conventions.

```
Name     = "Name" ":" relative-part
```

The relative-part production is defined in [RFC3986].

6. Entry Collection

Entry Collections are Collections that restrict their membership to Atom entries. This specification defines two serializations for Atom entries. Servers MUST provide both serializations.

1. Atom Entry Documents (application/atom+xml), [AtomFormat].

2. Atom Entry Documents wrapped by a SOAP envelope (application/soap+xml), .

Clients use the HTTP 'Accept' request header to indicate their preference [RFC2616]. If no 'Accept' header is present in the request, the server is free to choose any serialization. When an HTTP request contains a body, clients MUST include a 'Content-Type' header, and servers MUST accept both application/atom+xml and application/soap+xml message bodies.

6.1 Editing Entry Resources

Atom entries are edited by sending HTTP requests to an individual entry's URI. Servers can determine the processing necessary to interpret a request by examining the request's HTTP method and 'Content-Type' header.

If the request method is POST and the 'Content-Type' is application/soap+xml, the SOAP document MUST contain a Web-Method property . This specifcation defines two values for that property, PUT and DELETE.

Processing Client Requests

	GET	PUT	DELETE	POST
No Body	Read	x	Delete	x
Atom Body	x	Update	x	x
SOAP Body with Web-Method PUT	x	x	x	Update
SOAP Body with Web-Method DELETE	x	x	x	Delete

6.2 Role of Atom Entry Elements During Editing

The elements of an Atom Entry Document are either a 'Writable

Element' or a 'Round Trip Element'.

Writable Element - An element of an Atom Entry whose value is editable by the client and not enforced by the server.

Round Trip Element - An element of an Atom Entry whose value is enforced by the server and not editable by the client.

That categorization will determine the elements' disposition during editing.

Atom Entry Element	Property
atom:author	Writable
atom:category	Writable
atom:content	Writable
atom:contributor	Writable
atom:id	Round Trip
atom:link	Writable
atom:published	Writable
atom:source	Writable
atom:summary	Writable
atom:title	Writable
atom:updated	Round Trip

Table 2

7. Generic Collection

Generic Collections are Collections that do not have uniform restrictions on the representations of the member resources.

7.1 Editing Generic Resources

Member resources are edited by sending HTTP requests to an individual resource's URI. Servers can determine the processing necessary to interpret a request by examining the request's HTTP method and 'Content-Type' header.

Processing Client Requests

	GET	PUT	DELETE	POST
No Body	Read	x	Delete	x
Any Body	x	Update	x	x

8. Introspection

In order for authoring to commence, a client must first discover the capabilities and locations of collections offered.

8.1 Introspection Document

The Introspection Document describes "workspaces", which are server-defined groupings of collections. There is no requirement that servers support multiple workspaces, and a collection may appear in more than one workspace.

The Introspection Document has the media-type 'application/atomserv+xml', see Section 11

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://purl.org/atom/app#">
  <workspace title="Main Site" >
    <collection contents="entries" title="My Blog Entries"
      href="http://example.org/reilly/feed" />
    <collection contents="generic" title="Documents"
      href="http://example.org/reilly/pic" />
  </workspace>
  <workspace title="Side Bar Blog">
    <collection contents="entries" title="Entries"
      href="http://example.org/reilly/feed" />
    <collection contents="http://example.net/booklist" title="Books"
      href="http://example.org/reilly/books" />
  </workspace>
</service>
```

8.1.1 Element Definitions

8.1.1.1 The 'app:service' Element

The "service" element is the document element of a Service Document, acting as a container for service data associated with one or more workspaces.

```
appService       element app:service {
      ( appWorkspace*
        & anyElement* )
   }
```

The following child elements are defined by this specification:

- o app:service elements MAY contain any number of app:workspace elements.

8.1.1.2 The 'app:workspace' Element

The 'workspace' element element contains information elements about the collections of resources available for editing.

```
appWorkspace       element app:workspace {
      attribute title { text },
      ( appCollection*
        & anyElement* )
   }
```

The following attributes and child elements are defined by this specification:

- o app:workspace elements MUST contain a 'title' attribute, which conveys a human-readable name for the workspace

- o app:workspace elements MAY contain any number of app:collection elements.

8.1.1.3 The 'app:collection' Element

The 'app:collection' element describes collections and their member resources.

[[anchor19: We have a collection element that's different than the root element of the collection document. Messy. --R. Sayre]]

```
appCollection       element app:collection {
      attribute title { text },
      attribute contents { text },
      attribute href { text },
      anyElement*
   }
```

The following attributes are defined by this specification:

- o app:collection elements MUST contain a 'title' attribute, whose value conveys a human-readable name for the workspace

- o app:collection elements MAY contain a 'contents' attribute (Section 8.1.1.3.1). If it is not present, it's value is considered to be 'generic'.

- o app:collection elements MUST contain an 'href' attribute, whose value conveys the URI of the collection.

8.1.1.3.1 The 'contents' Attribute

The 'contents' attribute conveys the nature of a collection's member resources. This specification defines two initial values for the 'contents' attribute:

- o entry

- o generic

Extensibility for 'content' values is handled [[anchor20: Same as atom:link]].

8.1.1.3.1.1 entry

A value of 'entry' for the contents attribute indicates that the Collection is an Entry Collection (Section 6).

8.1.1.3.1.2 generic

A value of 'generic' for the contents attribute indicates that the Collection is a Generic Collection (Section 7).

8.2 Introspection Resource

To retrieve an Introspection Document, the client sends a GET request to its URI.

```
GET /service-desc HTTP/1.1
Host: example.org
User-Agent: Cosimo/1.0
Accept: application/atomserv+xml
```

The server responds to a GET request by returning an Introspection Document in the message body.

```
HTTP/1.1 200 OK
Date: Mon, 21 Mar 2005 19:20:19 GMT
Server: CountBasic/2.0
Last-Modified: Mon, 21 Mar 2005 19:17:26 GMT
ETag: "4c083-268-423f1dc6"
Content-Length: nnnn
Content-Type: application/atomserv+xml

<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://purl.org/atom/app#">
    ...
</service>
```

8.2.1 Discovery

[[anchor24: Add in desc of an HTML link element that points to the Introspection Resource, or add it to the autodisco draft]]

9. 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.

9.1 [@@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.

10. 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.

11. IANA Considerations

A Atom Collection Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: atomcoll+xml

Mandatory parameters: None.

Optional parameters:

"charset": This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], section 3.2.

Security considerations: As defined in this specification. [[anchor28: update upon publication]]

In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification. [[anchor29: update upon publication]]

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], section 3.2.

File extension: .atomcoll

Fragment identifiers: As specified for "application/xml" in [RFC3023], section 5.

Base URI: As specified in [RFC3023], section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: Joe Gregorio <joe@bitworking.org>

Intended usage: COMMON

Author/Change controller: IESG

An Atom Introspection Document, when serialized as XML 1.0, can be identified with the following media type:

MIME media type name: application

MIME subtype name: atomserv+xml

Mandatory parameters: None.

Optional parameters:

"charset": This parameter has identical semantics to the charset parameter of the "application/xml" media type as specified in [RFC3023].

Encoding considerations: Identical to those of "application/xml" as described in [RFC3023], section 3.2.

Security considerations: As defined in this specification. [[anchor30: update upon publication]]

In addition, as this media type uses the "+xml" convention, it shares the same security considerations as described in [RFC3023], section 10.

Interoperability considerations: There are no known interoperability issues.

Published specification: This specification. [[anchor31: update upon publication]]

Applications that use this media type: No known applications currently use this media type.

Additional information:

Magic number(s): As specified for "application/xml" in [RFC3023], section 3.2.

File extension: .atomsrv

Fragment identifiers: As specified for "application/xml" in [RFC3023], section 5.

Base URI: As specified in [RFC3023], section 6.

Macintosh File Type code: TEXT

Person and email address to contact for further information: Joe Gregorio <joe@bitworking.org>

Intended usage: COMMON

Author/Change controller: This specification's author(s). [[anchor32: update upon publication]]

12. References

12.1 Normative References

[AtomFormat]
Nottingham, M. and R. Sayre, "The Atom Syndication Format", work-in-progress, April 2005.

[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.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, January 2005.

[W3C.REC-soap12-part1-20030624]
Nielsen, H., Mendelsohn, N., Gudgin, M., Hadley, M., and J. Moreau, "SOAP Version 1.2 Part 1: Messaging Framework", W3C REC REC-soap12-part1-20030624, June 2003.

[W3C.REC-soap12-part2-20030624]
Nielsen, H., Hadley, M., Moreau, J., Mendelsohn, N., and M. Gudgin, "SOAP Version 1.2 Part 2: Adjuncts", W3C REC REC-soap12-part2-20030624, June 2003.

[W3C.REC-xml-20040204]
Yergeau, F., Paoli, J., Sperberg-McQueen, C., Bray, T.,

and E. Maler, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C REC REC-xml-20040204, February 2004.

12.2 Informative References

[W3C.REC-webarch-20041215]
Walsh, N. and I. Jacobs, "Architecture of the World Wide Web, Volume One", W3C REC REC-webarch-20041215, December 2004.

URIs

[1] <http://www.imc.org/atom-protocol/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)

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

Appendix A. Revision History

draft-ietf-atompub-protocol-04 - Add ladder diagrams, reorganize, add SOAP interactions

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.

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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