Network Working Group
Request for Comments: 5023
Category: Standards Track

J. Gregorio, Ed.
Google
B. de hOra, Ed.
NewBay Software

October 2007

The Atom Publishing Protocol

Status of This Memo

Abstract

The Atom Publishing Protocol (AtomPub) is an application-level protocol for publishing and editing Web resources. The protocol is based on HTTP transfer of Atom-formatted representations. The Atom format is documented in the Atom Syndication Format.

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 [REC-xml]. The protocol supports the creation of Web Resources and provides facilities for:

- Collections: Sets of Resources, which can be retrieved in whole or in part.
- Services: Discovery and description of Collections.
- Editing: Creating, editing, and deleting Resources.

The Atom Publishing Protocol is different from many contemporary protocols in that the server is given wide latitude in processing requests from clients. See Section 4.4 for more details.

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].

2.1. XML-Related Conventions

2.1.1. Referring to Information Items

Atom Protocol Document formats are specified in terms of the XML Information Set [REC-xml-infoset], serialized as XML 1.0 [REC-xml].

The Infoset terms "Element Information Item" and "Attribute Information Item" are shortened to "element" and "attribute" respectively. Therefore, when this specification uses the term "element", it is referring to an Element Information Item, and when it uses the term "attribute", it is referring to an Attribute Information Item.

2.1.2. RELAX NG Schema

Some sections of this specification are illustrated with fragments of a non-normative RELAX NG Compact schema [RNC]. However, the text of this specification provides the definition of conformance. Complete schemas appear in Appendix B.

2.1.3. Use of "xml:base" and "xml:lang"

XML elements defined by this specification MAY have an "xml:base" attribute [REC-xmlbase]. When xml:base is used, it serves the function described in Section 5.1.1 of URI Generic Syntax [RFC3986], by establishing the base URI (or IRI, Internationalized Resource Identifier [RFC3987]) for resolving relative references found within the scope of the "xml:base" attribute.

Any element defined by this specification MAY have an "xml:lang" attribute, whose content indicates the natural language for the element and its descendants. Requirements regarding the content and interpretation of "xml:lang" are specified in Section 2.12 of XML 1.0 [REC-xml].

3. Terminology

For convenience, this protocol can be referred to as the "Atom Protocol" or "AtomPub". The following terminology is used by this specification:

- URI - A Uniform Resource Identifier as defined in [RFC3986]. In this specification, the phrase "the URI of a document" is shorthand for "a URI which, when dereferenced, is expected to produce that document as a representation".

- IRI - An Internationalized Resource Identifier as defined in [RFC3987]. Before an IRI found in a document is used by HTTP, the IRI is first converted to a URI. See Section 4.1.

- Resource - A network-accessible data object or service identified by an IRI, as defined in [RFC2616]. See [REC-webarch] for further discussion on Resources.

- relation (or "relation of") - Refers to the "rel" attribute value of an atom:link element.

- Representation - An entity included with a request or response as defined in [RFC2616].

- Collection - A Resource that contains a set of Member Resources. Collections are represented as Atom Feeds. See Section 9.

- Member (or Member Resource) - A Resource whose IRI is listed in a Collection by an atom:link element with a relation of "edit" or "edit-media". See Section 9.1. The protocol defines two kinds of Members:
  - Entry Resource - Members of a Collection that are represented as Atom Entry Documents, as defined in [RFC4287].
  - Media Resource - Members of a Collection that have representations other than Atom Entry Documents.
- Media Link Entry - An Entry Resource that contains metadata about a Media Resource. See Section 9.6.
- Workspace - A named group of Collections. See Section 8.1.
- Service Document - A document that describes the location and capabilities of one or more Collections, grouped into Workspaces. See Section 8.
- Category Document - A document that describes the categories allowed in a Collection. See Section 7.

4. Protocol Model

The Atom Protocol specifies operations for publishing and editing Resources using HTTP. It uses Atom-formatted representations to describe the state and metadata of those Resources. It defines how Collections of Resources can be organized, and it specifies formats to support their discovery, grouping and categorization.

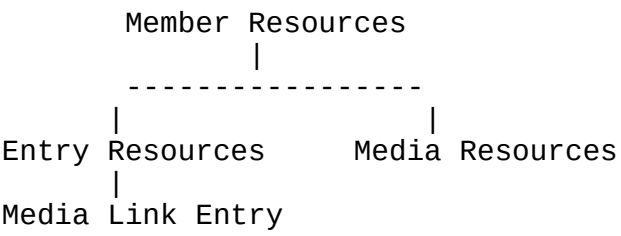
4.1. Identity and Naming

Atom Protocol documents allow the use of IRIs [RFC3987] as well as URIs [RFC3986] to identify Resources. Before an IRI in a document is used by HTTP, the IRI is first converted to a URI according to the procedure defined in Section 3.1 of [RFC3987]. In accordance with that specification, the conversion SHOULD be applied as late as possible. Conversion does not imply Resource creation -- the IRI and the URI into which it is converted identify the same Resource.

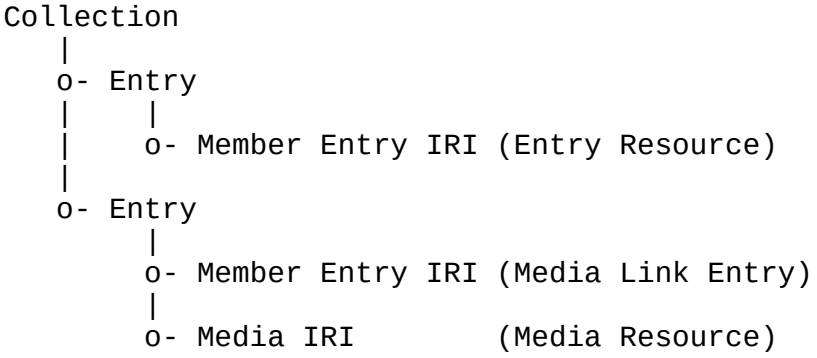
While the Atom Protocol specifies the formats of the representations that are exchanged and the actions that can be performed on the IRIs embedded in those representations, it does not constrain the form of the URIs that are used. HTTP [RFC2616] specifies that the URI space of each server is controlled by that server, and this protocol imposes no further constraints on that control.

4.2. Documents and Resource Classification

A Resource whose IRI is listed in a Collection is called a Member Resource. The protocol defines two kinds of Member Resources -- Entry Resources and Media Resources. Entry Resources are represented as Atom Entry Documents [RFC4287]. Media Resources can have representations in any media type. A Media Resource is described within a Collection using an Entry called a Media Link Entry. This diagram shows the classification of Resources within the Atom Protocol:

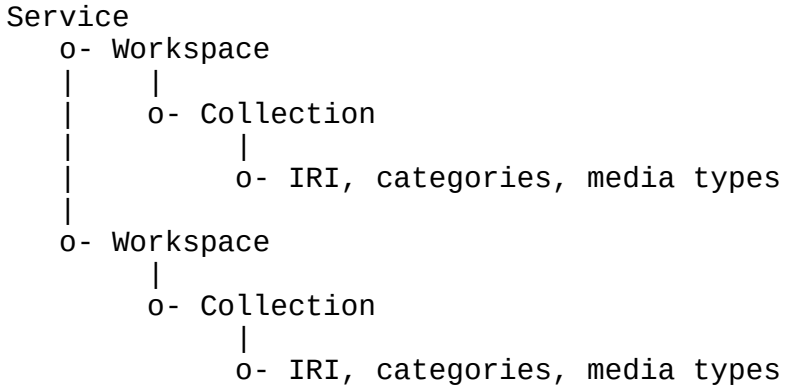


The Atom Protocol defines Collection Resources for managing and organizing both kinds of Member Resource. A Collection is represented by an Atom Feed Document. A Collection Feed's Entries contain the IRIs of, and metadata about, the Collection's Member Resources. A Collection Feed can contain any number of Entries, which might represent all the Members of the Collection, or an ordered subset of them (see Section 10.1). In the diagram of a Collection below, there are two Entries. The first contains the IRI of an Entry Resource. The second contains the IRIs of both a Media Resource and a Media Link Entry, which contains the metadata for that Media Resource:



The Atom Protocol does not make a distinction between Feeds used for Collections and other Atom Feeds. The only mechanism that this specification supplies for indicating that a Feed is a Collection Feed is the presence of the Feed's IRI in a Service Document.

Service Documents represent server-defined groups of Collections, and are used to initialize the process of creating and editing Resources. These groups of Collections are called Workspaces. Workspaces have names, but no IRIs, and no specified processing model. The Service Document can indicate which media types, and which categories, a Collection will accept. In the diagram below, there are two Workspaces each describing the IRIs, acceptable media types, and categories for a Collection:



4.3. Control and Publishing

The Atom Publishing Protocol uses HTTP methods to author Member Resources as follows:

- GET is used to retrieve a representation of a known Resource.
- POST is used to create a new, dynamically named, Resource. When the client submits non-Atom-Entry representations to a Collection for creation, two Resources are always created -- a Media Entry for the requested Resource, and a Media Link Entry for metadata about the Resource that will appear in the Collection.
- PUT is used to edit a known Resource. It is not used for Resource creation.
- DELETE is used to remove a known Resource.

The Atom Protocol only covers the creating, editing, and deleting of Entry and Media Resources. Other Resources could be created, edited, and deleted as the result of manipulating a Collection, but the number of those Resources, their media types, and effects of Atom Protocol operations on them are outside the scope of this specification.

Since all aspects of client-server interaction are defined in terms of HTTP, [RFC2616] should be consulted for any areas not covered in this specification.

4.4. Client Implementation Considerations

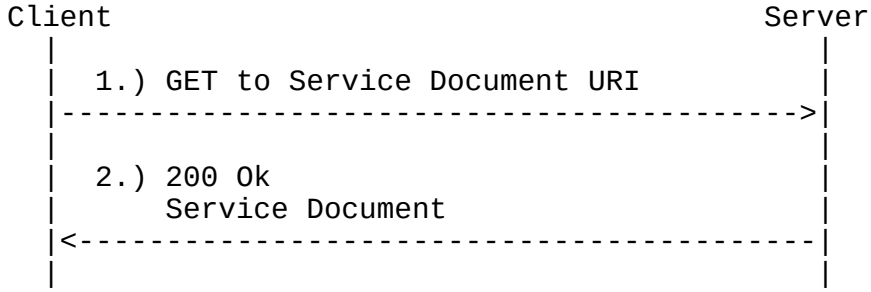
The Atom Protocol imposes few restrictions on the actions of servers. Unless a constraint is specified here, servers can be expected to vary in behavior, in particular around the manipulation of Atom Entries sent by clients. For example, although this specification only defines the expected behavior of Collections with respect to GET and POST, this does not imply that PUT, DELETE, PROPPATCH, and others are forbidden on Collection Resources -- only that this specification does not define what the server's response would be to those methods. Similarly, while some HTTP status codes are mentioned explicitly, clients ought to be prepared to handle any status code from a server. Servers can choose to accept, reject, delay, moderate, censor, reformat, translate, relocate, or re-categorize the content submitted to them. Only some of these choices are immediately relayed back to the client in responses to client requests; other choices may only become apparent later, in the feed or published entries. The same series of requests to two different publishing sites can result in a different series of HTTP responses, different resulting feeds, or different entry contents.

As a result, client software has to be written flexibly to accept what the server decides are the results of its submissions. Any server response or server content modification not explicitly forbidden by this specification or HTTP [RFC2616] is therefore allowed.

5. Protocol Operations

While specific HTTP status codes are shown in the interaction diagrams below, an AtomPub client should be prepared to handle any status code. For example, a PUT to a Member URI could result in the return of a "204 No Content" status code, which still indicates success.

5.1. Retrieving a Service Document

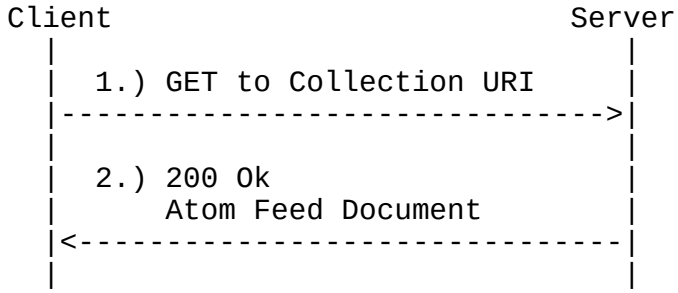


1. The client sends a GET request to the URI of the Service Document.

2. The server responds with a Service Document enumerating the IRIs of a group of Collections and the capabilities of those Collections supported by the server. The content of this document can vary based on aspects of the client request, including, but not limited to, authentication credentials.

5.2. Listing Collection Members

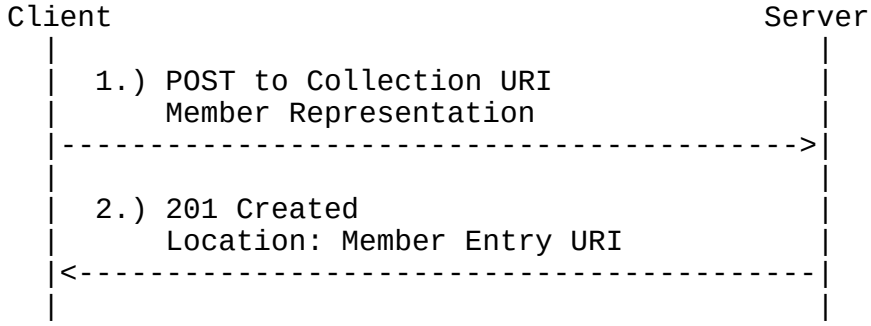
To list the Members of a Collection, the client sends a GET request to the URI of a Collection. An Atom Feed Document is returned whose Entries contain the IRIs of Member Resources. The returned Feed may describe all, or only a partial list, of the Members in a Collection (see Section 10).



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

2. The server responds with an Atom Feed Document containing the IRIs of the Collection Members.

5.3. Creating a Resource



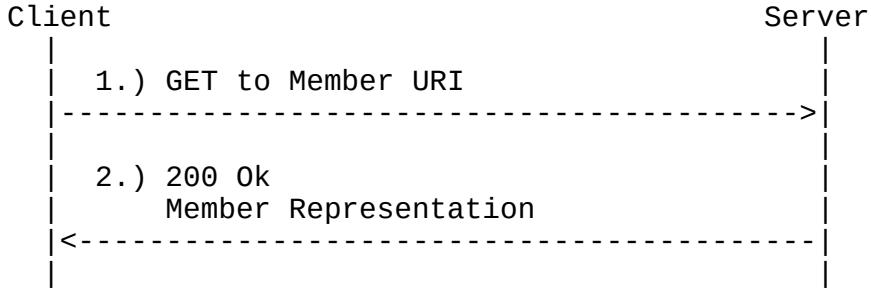
1. The client POSTs a representation of the Member to the URI of the Collection.

2. If the Member Resource was created successfully, the server responds with a status code of 201 and a Location header that contains the IRI of the newly created Entry Resource. Media Resources could have also been created and their IRIs can be found through the Entry Resource. See Section 9.6 for more details.

5.4. Editing a Resource

Once a Resource has been created and its Member URI is known, that URI can be used to retrieve, edit, and delete the Resource. Section 11 describes extensions to the Atom Syndication Format used in the Atom Protocol for editing purposes.

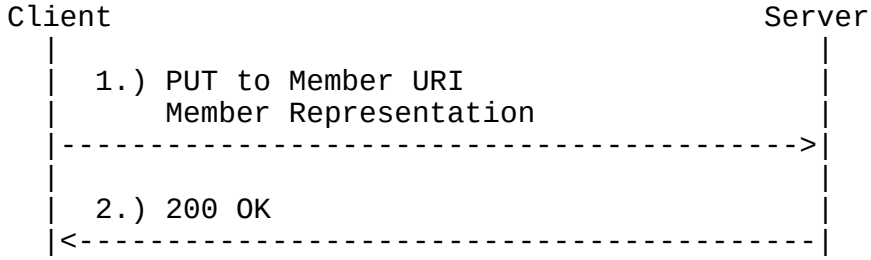
5.4.1. Retrieving a Resource



1. The client sends a GET request to the URI of a Member Resource to retrieve its representation.

2. The server responds with the representation of the Member Resource.

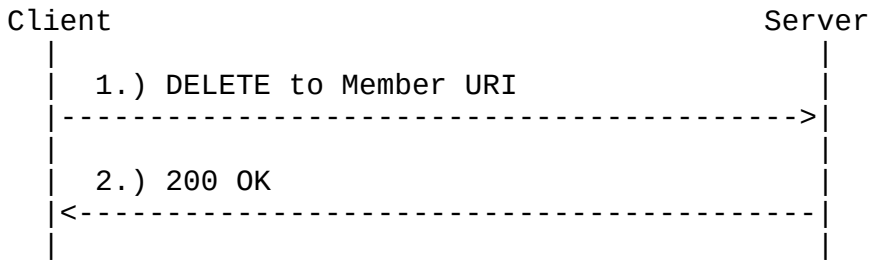
5.4.2. Editing a Resource



1. The client sends a PUT request to store a representation of a Member Resource.

2. If the request is successful, the server responds with a status code of 200.

5.4.3. Deleting a Resource



1. The client sends a DELETE request to the URI of a Member Resource.

2. If the deletion is successful, the server responds with a status code of 200.

A different approach is taken for deleting Media Resources; see Section 9.4 for details.

5.5. Use of HTTP Response Codes

The Atom Protocol uses the response status codes defined in HTTP to indicate the success or failure of an operation. Consult the HTTP specification [RFC2616] for detailed definitions of each status code.

Implementers are asked to note that according to the HTTP specification, HTTP 4xx and 5xx response entities SHOULD include a human-readable explanation of the error.

6. Protocol Documents

6.1. Document Types

This specification defines two kinds of documents -- Category Documents and Service Documents.

A Category Document (Section 7) contains lists of categories specified using the "atom:category" element from the Atom Syndication Format (see Section 4.2.2 of [RFC4287]).

A Service Document (Section 8) groups available Collections into Workspaces.

The namespace name [REC-xml-names] for either kind of document is:

    http://www.w3.org/2007/app

Atom Publishing Protocol XML Documents MUST be "namespace-well-formed" as specified in Section 7 of [REC-xml-names].

This specification uses the prefix "app:" for the namespace name. The prefix "atom:" is used for "http://www.w3.org/2005/Atom", the namespace name of the Atom Syndication Format [RFC4287]. These namespace prefixes are not semantically significant.

This specification does not define any DTDs for Atom Protocol formats, and hence does not require them to be "valid" in the sense used by [REC-xml].

6.2. Document Extensibility

Unrecognized markup in an Atom Publishing Protocol document is considered "foreign markup" as defined in Section 6 of the Atom Syndication Format [RFC4287]. Foreign markup can be used anywhere within a Category or Service Document unless it is explicitly forbidden. Processors that encounter foreign markup MUST NOT stop processing and MUST NOT signal an error. Clients SHOULD preserve foreign markup when transmitting such documents.

The namespace name "http://www.w3.org/2007/app" is reserved for forward-compatible revisions of the Category and Service Document types. This does not exclude the addition of elements and attributes that might not be recognized by processors conformant to this specification. Such unrecognized markup from the "http://www.w3.org/2007/app" namespace MUST be treated as foreign markup.

7. Category Documents

Category Documents contain lists of categories described using the "atom:category" element from the Atom Syndication Format [RFC4287]. Categories can also appear in Service Documents, where they indicate the categories allowed in a Collection (see Section 8.3.6).

Category Documents are identified with the "application/atomcat+xml" media type (see Section 16.1).

7.1. Example

```
<?xml version="1.0" ?>
<app:categories
    xmlns:app="http://www.w3.org/2007/app"
    xmlns:atom="http://www.w3.org/2005/Atom"
    fixed="yes" scheme="http://example.com/cats/big3">
  <atom:category term="animal" />
  <atom:category term="vegetable" />
  <atom:category term="mineral" />
</app:categories>
```

This Category Document contains atom:category elements, with the terms 'animal', 'vegetable', and 'mineral'. None of the categories use the "label" attribute defined in [RFC4287]. They all inherit the "http://example.com/cats/big3" "scheme" attribute declared on the app:categories element. Therefore if the 'mineral' category were to appear in an Atom Entry or Feed Document, it would appear as:

```
<atom:category scheme="http://example.com/cats/big3" term="mineral"/>
```

7.2. Element Definitions

7.2.1. The "app:categories" Element

The root of a Category Document is the "app:categories" element. An app:categories element can contain zero or more atom:category elements from the Atom Syndication Format [RFC4287] namespace ("http://www.w3.org/2005/Atom").

An atom:category child element that has no "scheme" attribute inherits the attribute from its app:categories parent. An atom: category child element with an existing "scheme" attribute does not inherit the "scheme" value of its app:categories parent element.

```
atomCategory =
    element atom:category {
       atomCommonAttributes,
       attribute term { text },
       attribute scheme { atomURI }?,
       attribute label { text }?,
       undefinedContent
    }

appInlineCategories =
    element app:categories {
        attribute fixed { "yes" | "no" }?,
        attribute scheme { atomURI }?,
        (atomCategory*,
        undefinedContent)
    }

appOutOfLineCategories =
    element app:categories {
        attribute href { atomURI },
        undefinedContent
    }

appCategories = appInlineCategories | appOutOfLineCategories
```

7.2.1.1. Attributes of "app:categories"

The app:categories element can contain a "fixed" attribute, with a value of either "yes" or "no", indicating whether the list of categories is a fixed or an open set. The absence of the "fixed" attribute is equivalent to the presence of a "fixed" attribute with a value of "no".

Alternatively, the app:categories element MAY contain an "href" attribute, whose value MUST be an IRI reference identifying a Category Document. If the "href" attribute is provided, the app:categories element MUST be empty and MUST NOT have the "fixed" or "scheme" attributes.

8. Service Documents

For authoring to commence, a client needs to discover the capabilities and locations of the available Collections. Service Documents are designed to support this discovery process.

How Service Documents are discovered is not defined in this specification.

Service Documents are identified with the "application/atomsvc+xml" media type (see Section 16.2).

8.1. Workspaces

A Service Document groups Collections into Workspaces. Operations on Workspaces, such as creation or deletion, are not defined by this specification. This specification assigns no meaning to Workspaces; that is, a Workspace does not imply any specific processing assumptions.

There is no requirement that a server support multiple Workspaces. In addition, a Collection MAY appear in more than one Workspace.

8.2. Example

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://www.w3.org/2007/app"
         xmlns:atom="http://www.w3.org/2005/Atom">
  <workspace>
    <atom:title>Main Site</atom:title>
    <collection
        href="http://example.org/blog/main" >
      <atom:title>My Blog Entries</atom:title>
      <categories
         href="http://example.com/cats/forMain.cats" />
    </collection>
    <collection
        href="http://example.org/blog/pic" >
      <atom:title>Pictures</atom:title>
      <accept>image/png</accept>
      <accept>image/jpeg</accept>
      <accept>image/gif</accept>
    </collection>
  </workspace>
  <workspace>
    <atom:title>Sidebar Blog</atom:title>
    <collection
        href="http://example.org/sidebar/list" >
      <atom:title>Remaindered Links</atom:title>
      <accept>application/atom+xml;type=entry</accept>
      <categories fixed="yes">
        <atom:category
          scheme="http://example.org/extra-cats/"
          term="joke" />
        <atom:category
          scheme="http://example.org/extra-cats/"
          term="serious" />
```

```
      </categories>
    </collection>
  </workspace>
</service>
```

The Service Document above describes two Workspaces. The first Workspace is called "Main Site", and has two Collections called "My Blog Entries" and "Pictures", whose IRIs are "http://example.org/blog/main" and "http://example.org/blog/pic" respectively. The "Pictures" Collection includes three "accept" elements indicating the types of image files the client can send to the Collection to create new Media Resources (entries associated with Media Resources are discussed in Section 9.6).

The second Workspace is called "Sidebar Blog" and has a single Collection called "Remaindered Links" whose IRI is "http://example.org/sidebar/list". The Collection has an "accept" element whose content is "application/atom+xml;type=entry", indicating it will accept Atom Entries from a client.

Within each of the two Entry Collections, the "categories" element provides a list of available categories for Member Entries. In the "My Blog Entries" Collection, the list of available categories is available through the "href" attribute. The "Sidebar Blog" Collection provides a category list within the Service Document, but states the list is fixed, signaling a request from the server that Entries be POSTed using only those two categories.

8.3. Element Definitions

8.3.1. The "app:service" Element

The root of a Service Document is the "app:service" element.

The app:service element is the container for service information associated with one or more Workspaces. An app:service element MUST contain one or more app:workspace elements.

```
namespace app = "http://www.w3.org/2007/app"
start = appService

appService =
   element app:service {
      appCommonAttributes,
      ( appWorkspace+
        & extensionElement* )
   }
```

8.3.2. The "app:workspace" Element

Workspaces are server-defined groups of Collections. The "app:workspace" element contains zero or more app:collection elements describing the Collections of Resources available for editing.

```
appWorkspace =
   element app:workspace {
      appCommonAttributes,
      ( atomTitle
        & appCollection*
        & extensionSansTitleElement* )
   }

atomTitle = element atom:title { atomTextConstruct }
```

8.3.2.1. The "atom:title" Element

The app:workspace element MUST contain one "atom:title" element (as defined in [RFC4287]), giving a human-readable title for the Workspace.

8.3.3. The "app:collection" Element

The "app:collection" element describes a Collection. The app:collection element MUST contain one atom:title element.

The app:collection element MAY contain any number of app:accept elements, indicating the types of representations accepted by the Collection. The order of such elements is not significant.

The app:collection element MAY contain any number of app:categories elements.

```
appCollection =
   element app:collection {
      appCommonAttributes,
      attribute href { atomURI  },
      ( atomTitle
        & appAccept*
        & appCategories*
        & extensionSansTitleElement* )
   }
```

8.3.3.1. The "href" Attribute

The app:collection element MUST contain an "href" attribute, whose value gives the IRI of the Collection.

8.3.3.2. The "atom:title" Element

The "atom:title" element is defined in [RFC4287] and gives a human-readable title for the Collection.

8.3.4. The "app:accept" Element

The content of an "app:accept" element value is a media range as defined in [RFC2616]. The media range specifies a type of representation that can be POSTed to a Collection.

The app:accept element is similar to the HTTP Accept request-header [RFC2616]. Media type parameters are allowed within app:accept, but app:accept has no notion of preference -- "accept-params" or "q" arguments, as specified in Section 14.1 of [RFC2616] are not significant.

White space (as defined in [REC-xml]) around the app:accept element's media range is insignificant and MUST be ignored.

A value of "application/atom+xml;type=entry" MAY appear in any app:accept list of media ranges and indicates that Atom Entry Documents can be POSTed to the Collection. If no app:accept element is present, clients SHOULD treat this as equivalent to an app:accept element with the content "application/atom+xml;type=entry".

If one app:accept element exists and is empty, clients SHOULD assume that the Collection does not support the creation of new Entries.

```
appAccept =
   element app:accept {
         appCommonAttributes,
         ( text? )
   }
```

8.3.5. Usage in Atom Feed Documents

The app:collection element MAY appear as a child of an atom:feed or atom:source element in an Atom Feed Document. Its content identifies a Collection by which new Entries can be added to appear in the feed. When it appears in an atom:feed or atom:source element, the app:collection element is considered foreign markup as defined in Section 6 of [RFC4287].

8.3.6. The "app:categories" Element

The "app:categories" element provides a list of the categories that can be applied to the members of a Collection. See Section 7.2.1 for the detailed definition of app:categories.

The server MAY reject attempts to create or store members whose categories are not present in its categories list. A Collection that indicates the category set is open SHOULD NOT reject otherwise acceptable members whose categories are not in its categories list. The absence of an app:categories element means that the category handling of the Collection is unspecified. A "fixed" category list that contains zero categories indicates the Collection does not accept category data.

9. Creating and Editing Resources

9.1. Member URIs

The Member URI allows clients to retrieve, edit, and delete a Member Resource using HTTP's GET, PUT, and DELETE methods. Entry Resources are represented as Atom Entry documents.

Member URIs appear in two places. They are returned in a Location header after successful Resource creation using POST, as described in Section 9.2 below. They can also appear in a Collection Feed's Entries, as atom:link elements with a link relation of "edit".

A Member Entry SHOULD contain such an atom:link element with a link relation of "edit", which indicates the Member URI.

9.2. Creating Resources with POST

To add members to a Collection, clients send POST requests to the URI of the Collection.

Successful member creation is indicated with a 201 ("Created") response code. When the Collection responds with a status code of 201, it SHOULD also return a response body, which MUST be an Atom Entry Document representing the newly created Resource. Since the server is free to alter the POSTed Entry, for example, by changing the content of the atom:id element, returning the Entry can be useful to the client, enabling it to correlate the client and server views of the new Entry.

When a Member Resource is created, its Member Entry URI MUST be returned in a Location header in the Collection's response.

If the creation request contained an Atom Entry Document, and the subsequent response from the server contains a Content-Location header that matches the Location header character-for-character, then the client is authorized to interpret the response entity as being a complete representation of the newly created Entry. Without a matching Content-Location header, the client MUST NOT assume the returned entity is a complete representation of the created Resource.

The request body sent with the POST need not be an Atom Entry. For example, it might be a picture or a movie. Collections MAY return a response with a status code of 415 ("Unsupported Media Type") to indicate that the media type of the POSTed entity is not allowed or supported by the Collection. For a discussion of the issues in creating such content, see Section 9.6.

9.2.1. Example

Below, the client sends a POST request containing an Atom Entry representation using the URI of the Collection:

```
POST /edit/ HTTP/1.1
Host: example.org
User-Agent: Thingio/1.0
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
Slug: First Post

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>
  <content>Some text.</content>
</entry>
```

The server signals a successful creation with a status code of 201. The response includes a Location header indicating the Member Entry URI of the Atom Entry, and a representation of that Entry in the body of the response.

```
HTTP/1.1 201 Created
Date: Fri, 7 Oct 2005 17:17:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
Location: http://example.org/edit/first-post.atom
ETag: "c180de84f991g8"
```

```
<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>
  <content>Some text.</content>
  <link rel="edit"
      href="http://example.org/edit/first-post.atom"/>
</entry>
```

The Entry created and returned by the Collection might not match the Entry POSTed by the client. A server MAY change the values of various elements in the Entry, such as the atom:id, atom:updated, and atom:author values, and MAY choose to remove or add other elements and attributes, or change element content and attribute values.

9.3. Editing Resources with PUT

To edit a Member Resource, a client sends a PUT request to its Member URI, as specified in [RFC2616].

To avoid unintentional loss of data when editing Member Entries or Media Link Entries, an Atom Protocol client SHOULD preserve all metadata that has not been intentionally modified, including unknown foreign markup as defined in Section 6 of [RFC4287].

9.4. Deleting Resources with DELETE

To delete a Member Resource, a client sends a DELETE request to its Member URI, as specified in [RFC2616]. The deletion of a Media Link Entry SHOULD result in the deletion of the corresponding Media Resource.

9.5. Caching and Entity Tags

Implementers are advised to pay attention to cache controls and to make use of the mechanisms available in HTTP when editing Resources, in particular, entity-tags as outlined in [NOTE-detect-lost-update]. Clients are not assured to receive the most recent representations of Collection Members using GET if the server is authorizing intermediaries to cache them.

9.5.1. Example

Below, the client creates a Member Entry using POST:

```
POST /myblog/entries HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
Slug: First Post

<?xml version="1.0" ?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2007-02-123T17:09:02Z</updated>
  <author><name>Captain Lansing</name></author>
  <content>It's something moving... solid metal</content>
</entry>
```

The server signals a successful creation with a status code of 201, and returns an ETag header in the response. Because, in this case, the server returned a Content-Location header and Location header with the same value, the returned Entry representation can be understood to be a complete representation of the newly created Entry (see Section 9.2).

```
HTTP/1.1 201 Created
Date: Fri, 23 Feb 2007 21:17:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry
Location: http://example.org/edit/first-post.atom
Content-Location: http://example.org/edit/first-post.atom
ETag: "e180ee84f0671b1"

<?xml version="1.0" ?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2007-02-123T17:09:02Z</updated>
  <author><name>Captain Lansing</name></author>
  <content>It's something moving... solid metal</content>
</entry>
```

The client can, if it wishes, use the returned ETag value to later construct a "Conditional GET" as defined in [RFC2616]. In this case, prior to editing, the client sends the ETag value for the Member using the If-None-Match header.

```
GET /edit/first-post.atom HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
If-None-Match: "e180ee84f0671b1"
```

If the Entry has not been modified, the response will be a status code of 304 ("Not Modified"). This allows the client to determine whether it still has the most recent representation of the Entry at the time of editing.

```
HTTP/1.1 304 Not Modified
Date: Sat, 24 Feb 2007 13:17:11 GMT
```

After editing, the client can PUT the Entry and send the ETag entity value in an If-Match header, informing the server to accept the entry on the condition that the entity value sent still matches the server's.

```
PUT /edit/first-post.atom HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
If-Match: "e180ee84f0671b1"

<?xml version="1.0" ?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2007-02-24T16:34:06Z</updated>
  <author><name>Captain Lansing</name></author>
  <content>Update: it's a hoax!</content>
</entry>
```

The server however has since received a more recent copy than the client's, and it responds with a status code of 412 ("Precondition Failed").

```
HTTP/1.1 412 Precondition Failed
Date: Sat, 24 Feb 2007 16:34:11 GMT
```

This informs the client that the server has a more recent version of the Entry and will not allow the sent entity to be stored.

9.6. Media Resources and Media Link Entries

A client can POST Media Resources as well as Entry Resources to a Collection. If a server accepts such a request, then it MUST create two new Resources -- one that corresponds to the entity sent in the request, called the Media Resource, and an associated Member Entry, called the Media Link Entry. Media Link Entries are represented as Atom Entries, and appear in the Collection.

The Media Link Entry contains the metadata and IRI of the (perhaps non-textual) Media Resource. The Media Link Entry thus makes the metadata about the Media Resource separately available for retrieval and alteration.

The server can signal the media types it will accept using the app:accept element in the Service Document, as specified in Section 8.3.4.

Successful responses to creation requests MUST include the URI of the Media Link Entry in the Location header. The Media Link Entry SHOULD contain an atom:link element with a link relation of "edit-media" that contains the Media Resource IRI. The Media Link Entry MUST have an atom:content element with a "src" attribute. The value of the "src" attribute is an IRI for the newly created Media Resource. It is OPTIONAL that the IRI of the "src" attribute on the atom:content element be the same as the Media Resource IRI. For example, the "src" attribute value might instead be a link into a static cache or content distribution network and not the Media Resource IRI.

Implementers are asked to note that [RFC4287] specifies that Atom Entries MUST contain an atom:summary element. Thus, upon successful creation of a Media Link Entry, a server MAY choose to populate the atom:summary element (as well as any other mandatory elements such as atom:id, atom:author, and atom:title) with content derived from the POSTed entity or from any other source. A server might not allow a client to modify the server-selected values for these elements.

For Resource creation, this specification only defines cases where the POST body has an Atom Entry entity declared as an Atom media type ("application/atom+xml"), or a non-Atom entity declared as a non-Atom media type. When a client is POSTing an Atom Entry to a Collection, it may use a media type of either "application/atom+xml" or "application/atom +xml;type=entry". This specification does not specify any request semantics or server behavior in the case where the POSTed media type is "application/atom+xml" but the body is something other than an Atom Entry. In particular, what happens on POSTing an Atom Feed Document to a Collection using the "application/atom+xml" media type is undefined.

The Atom Protocol does not specify a means to create multiple representations of the same Resource (for example, a PNG and a JPG of the same image) either on creation or editing.

9.6.1. Examples

Below, the client sends a POST request containing a PNG image to the URI of a Collection that accepts PNG images:

```
POST /edit/ HTTP/1.1
Host: media.example.org
Content-Type: image/png
Slug: The Beach
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

...binary data...
```

The server signals a successful creation with a status code of 201. The response includes a Location header indicating the Member URI of the Media Link Entry and a representation of that entry in the body of the response. The Media Link Entry includes a content element with a "src" attribute. It also contains a link with a link relation of "edit-media", specifying the IRI to be used for modifying the Media Resource.

```
HTTP/1.1 201 Created
Date: Fri, 7 Oct 2005 17:17:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
Location: http://example.org/media/edit/the_beach.atom

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>The Beach</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2005-10-07T17:17:08Z</updated>
  <author><name>Daffy</name></author>
  <summary type="text" />
  <content type="image/png"
     src="http://media.example.org/the_beach.png"/>
  <link rel="edit-media"
     href="http://media.example.org/edit/the_beach.png" />
  <link rel="edit"
     href="http://example.org/media/edit/the_beach.atom" />
</entry>
```

Later, the client sends a PUT request containing the new PNG using the URI indicated in the Media Link Entry's "edit-media" link:

```
PUT /edit/the_beach.png HTTP/1.1
Host: media.example.org
Content-Type: image/png
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

...binary data...
```

The server signals a successful edit with a status code of 200.

```
HTTP/1.1 200 Ok
Date: Fri, 8 Oct 2006 17:17:11 GMT
```

The client can edit the metadata for the picture. First GET the Media Link Entry:

```
GET /media/edit/the_beach.atom HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
```

The Media Link Entry is returned.

```
HTTP/1.1 200 Ok
Date: Fri, 7 Oct 2005 17:18:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
ETag: "c181bb840673b5"

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>The Beach</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2005-10-07T17:17:08Z</updated>
  <author><name>Daffy</name></author>
  <summary type="text" />
  <content type="image/png"
     src="http://media.example.org/the_beach.png"/>
  <link rel="edit-media"
     href="http://media.example.org/edit/the_beach.png" />
  <link rel="edit"
     href="http://example.org/media/edit/the_beach.atom" />
</entry>
```

The metadata can be updated, in this case to add a summary, and then PUT back to the server.

```
PUT /media/edit/the_beach.atom HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
If-Match: "c181bb840673b5"

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>The Beach</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2005-10-07T17:17:08Z</updated>
  <author><name>Daffy</name></author>
  <summary type="text">
      A nice sunset picture over the water.
  </summary>
  <content type="image/png"
     src="http://media.example.org/the_beach.png"/>
  <link rel="edit-media"
     href="http://media.example.org/edit/the_beach.png" />
  <link rel="edit"
     href="http://example.org/media/edit/the_beach.atom" />
</entry>
```

The update was successful.

```
HTTP/1.1 200 Ok
Date: Fri, 7 Oct 2005 17:19:11 GMT
Content-Length: 0
```

Multiple Media Resources can be added to the Collection.

```
POST /edit/ HTTP/1.1
Host: media.example.org
Content-Type: image/png
Slug: The Pier
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

...binary data...
```

The Resource is created successfully.

```
HTTP/1.1 201 Created
Date: Fri, 7 Oct 2005 17:17:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
Location: http://example.org/media/edit/the_pier.atom
```

```
<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>The Pier</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efe6b</id>
  <updated>2005-10-07T17:26:43Z</updated>
  <author><name>Daffy</name></author>
  <summary type="text" />
  <content type="image/png"
     src="http://media.example.org/the_pier.png"/>
  <link rel="edit-media"
     href="http://media.example.org/edit/the_pier.png" />
  <link rel="edit"
     href="http://example.org/media/edit/the_pier.atom" />
</entry>
```

The client can now create a new Atom Entry in the blog Entry Collection that references the two newly created Media Resources.

```
POST /blog/ HTTP/1.1
Host: example.org
Content-Type: application/atom+xml;type=entry
Slug: A day at the beach
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>A fun day at the beach</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6b</id>
  <updated>2005-10-07T17:40:02Z</updated>
  <author><name>Daffy</name></author>
  <content type="xhtml">
      <xhtml:div xmlns:xhtml="http://www.w3.org/1999/xhtml">
          <xhtml:p>We had a good day at the beach.
              <xhtml:img alt="the beach"
                  src="http://media.example.org/the_beach.png"/>
          </xhtml:p>
          <xhtml:p>Later we walked down to the pier.
              <xhtml:img  alt="the pier"
                  src="http://media.example.org/the_pier.png"/>
          </xhtml:p>
      </xhtml:div>
  </content>
</entry>
```

The Resource is created successfully.

```
HTTP/1.1 200 Ok
Date: Fri, 7 Oct 2005 17:20:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
Location: http://example.org/blog/atom/a-day-at-the-beach.atom

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>A fun day at the beach</title>
  <id>http://example.org/blog/a-day-at-the-beach.xhtml</id>
  <updated>2005-10-07T17:43:07Z</updated>
  <author><name>Daffy</name></author>
  <content type="xhtml">
      <xhtml:div xmlns:xhtml="http://www.w3.org/1999/xhtml">
          <xhtml:p>We had a good day at the beach.
              <xhtml:img alt="the beach"
                 src="http://media.example.org/the_beach.png"/>
          </xhtml:p>
          <xhtml:p>Later we walked down to the pier.
              <xhtml:img alt="the pier"
                 src="http://media.example.org/the_pier.png"/>
          </xhtml:p>
      </xhtml:div>
  </content>
  <link rel="edit"
    href="http://example.org/blog/edit/a-day-at-the-beach.atom"/>
  <link rel="alternate" type="text/html"
    href="http://example.org/blog/a-day-at-the-beach.html"/>
</entry>
```

Note that the returned Entry contains a link with a relation of "alternate" that points to the associated HTML page that was created -- this is not required by this specification, but is included to show the kinds of changes a server can make to an Entry.

9.7. The Slug Header

Slug is an HTTP entity-header whose presence in a POST to a Collection constitutes a request by the client to use the header's value as part of any URIs that would normally be used to retrieve the to-be-created Entry or Media Resources.

Servers MAY use the value of the Slug header when creating the Member URI of the newly created Resource, for instance, by using some or all of the words in the value for the last URI segment. Servers MAY also use the value when creating the atom:id, or as the title of a Media Link Entry (see Section 9.6).

Servers MAY choose to ignore the Slug entity-header. Servers MAY alter the header value before using it. For instance, a server might filter out some characters or replace accented letters with non-accented ones, replace spaces with underscores, change case, and so on.

9.7.1. Slug Header Syntax

The syntax of the Slug header is defined using the augmented BNF syntax defined in Section 2.1 of [RFC2616]:

```
LWS      = <defined in Section 2.2 of [RFC2616]>
slugtext = %x20-7E | LWS
Slug     = "Slug" ":" *slugtext
```

The field value is the percent-encoded value of the UTF-8 encoding of the character sequence to be included (see Section 2.1 of [RFC3986] for the definition of percent encoding, and [RFC3629] for the definition of the UTF-8 encoding).

Implementation note: to produce the field value from a character sequence, first encode it using the UTF-8 encoding, then encode all octets outside the ranges %20-24 and %26-7E using percent encoding (%25 is the ASCII encoding of "%", thus it needs to be escaped). To consume the field value, first reverse the percent encoding, then run the resulting octet sequence through a UTF-8 decoding process.

9.7.2. Example

Here is an example of the Slug header that uses percent-encoding to represent the Unicode character U+00E8 (LATIN SMALL LETTER E WITH GRAVE):

```
POST /myblog/entries HTTP/1.1
Host: example.org
Content-Type: image/png
Slug: The Beach at S%C3%A8te
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

...binary data...
```

See Section 9.2.1 for an example of the Slug header applied to the creation of an Entry Resource.

10. Listing Collections

Collection Resources MUST provide representations in the form of Atom Feed Documents whose Entries contain the IRIs of the Members in the Collection. No distinction is made between Collection Feeds and other kinds of Feeds -- a Feed might act both as a 'public' feed for subscription purposes and as a Collection Feed.

Each Entry in the Feed Document SHOULD have an atom:link element with a relation of "edit" (see Section 11.1).

The Entries in the returned Atom Feed SHOULD be ordered by their "app:edited" property, with the most recently edited Entries coming first in the document order. The app:edited value is not equivalent to the HTTP Last-Modified header and cannot be used to determine the freshness of cached responses.

Clients MUST NOT assume that an Atom Entry returned in the Feed is a full representation of an Entry Resource and SHOULD perform a GET on the URI of the Member Entry before editing it. See Section 9.5 for a discussion on the implications of cache control directives when obtaining entries.

10.1. Collection Partial Lists

Collections can contain large numbers of Resources. A client such as a web spider or web browser might be overwhelmed if the response to a GET contained every Entry in a Collection -- in turn the server might also waste bandwidth and processing time on generating a response that cannot be handled. For this reason, servers MAY respond to Collection GET requests with a Feed Document containing a partial list of the Collection's members, and a link to the next partial list feed, if it exists. The first such partial list returned MUST contain the most recently edited member Resources and MUST have an atom:link with a "next" relation whose "href" value is the URI of the next partial list of the Collection. This next partial list will contain the next most recently edited set of Member Resources (and an atom:link to the following partial list if it exists).

In addition to the "next" relation, partial list feeds MAY contain link elements with "rel" attribute values of "previous", "first", and "last", that can be used to navigate through the complete set of entries in the Collection.

For instance, suppose a client is supplied the URI "http://example.org/entries/go" of a Collection of Member Entries, where the server as a matter of policy avoids generating Feed Documents containing more than 10 Entries. The Atom Feed Document

for the Collection will then represent the first partial list of a set of 10 linked Feed Documents. The "first" relation references the initial Feed Document in the set and the "last" relation references the final Feed Document in the set. Within each document, the "previous" and "next" link relations reference the preceding and subsequent documents.

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <link rel="first"
        href="http://example.org/entries/go" />
  <link rel="next"
        href="http://example.org/entries/2" />
  <link rel="last"
        href="http://example.org/entries/10" />
  ...
</feed>
```

The "previous" and "next" link elements for the partial list feed located at "http://example.org/entries/2" would look like this:

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <link rel="first"
        href="http://example.org/entries/go" />
  <link rel="previous"
        href="http://example.org/entries/go" />
  <link rel="next"
        href="http://example.org/entries/3" />
  <link rel="last"
        href="http://example.org/entries/10" />
  ...
</feed>
```

10.2. The "app:edited" Element

The "app:edited" element is a Date construct (as defined by [RFC4287]), whose content indicates the last time an Entry was edited. If the entry has not been edited yet, the content indicates the time it was created. Atom Entry elements in Collection Documents SHOULD contain one app:edited element, and MUST NOT contain more than one.

```
appEdited = element app:edited ( atomDateConstruct )
```

The server SHOULD change the value of this element every time an Entry Resource or an associated Media Resource has been edited.

11. Atom Format Link Relation Extensions

11.1. The "edit" Link Relation

This specification adds the value "edit" to the Atom Registry of Link Relations (see Section 7.1 of [RFC4287]). The value of "edit" specifies that the value of the href attribute is the IRI of an editable Member Entry. When appearing within an atom:entry, the href IRI can be used to retrieve, update, and delete the Resource represented by that Entry. An atom:entry MUST NOT contain more than one "edit" link relation.

11.2. The "edit-media" Link Relation

This specification adds the value "edit-media" to the Atom Registry of Link Relations (see Section 7.1 of [RFC4287]). When appearing within an atom:entry, the value of the href attribute is an IRI that can be used to modify a Media Resource associated with that Entry.

An atom:entry element MAY contain zero or more "edit-media" link relations. An atom:entry MUST NOT contain more than one atom:link element with a "rel" attribute value of "edit-media" that has the same "type" and "hreflang" attribute values. All "edit-media" link relations in the same Entry reference the same Resource. If a client encounters multiple "edit-media" link relations in an Entry then it SHOULD choose a link based on the client preferences for "type" and "hreflang". If a client encounters multiple "edit-media" link relations in an Entry and has no preference based on the "type" and "hreflang" attributes then the client SHOULD pick the first "edit-media" link relation in document order.

12. The Atom Format Type Parameter

The Atom Syndication Format [RFC4287] defines the "application/atom+xml" media type to identify both Atom Feed and Atom Entry Documents. Implementation experience has demonstrated that Atom Feed and Entry Documents can have different processing models and that there are situations where they need to be differentiated. This specification defines a "type" parameter used to differentiate the two types of Atom documents.

12.1. The "type" parameter

This specification defines a new "type" parameter for use with the "application/atom+xml" media type. The "type" parameter has a value of "entry" or "feed".

Neither the parameter name nor its value are case sensitive.

The value "entry" indicates that the media type identifies an Atom Entry Document. The root element of the document MUST be atom:entry.

The value "feed" indicates that the media type identifies an Atom Feed Document. The root element of the document MUST be atom:feed.

If not specified, the type is assumed to be unspecified, requiring Atom processors to examine the root element to determine the type of Atom document.

12.1.1. Conformance

New specifications MAY require that the "type" parameter be used to identify the Atom Document type. Producers of Atom Entry Documents SHOULD use the "type" parameter regardless of whether or not it is mandatory. Producers of Atom Feed Documents MAY use the parameter.

Atom processors that do not recognize the "type" parameter MUST ignore its value and examine the root element to determine the document type.

Atom processors that do recognize the "type" parameter SHOULD detect and report inconsistencies between the parameter's value and the actual type of the document's root element.

13. Atom Publishing Controls

This specification defines an Atom Format Structured Extension, as defined in Section 6 of [RFC4287], for publishing control within the "http://www.w3.org/2007/app" namespace.

13.1. The "app:control" Element

```
namespace app = "http://www.w3.org/2007/app"

 pubControl =
    element app:control {
    atomCommonAttributes,
    pubDraft?
    & extensionElement
 }

 pubDraft =
   element app:draft { "yes" | "no" }
```

The "app:control" element MAY appear as a child of an atom:entry that is being created or updated via the Atom Publishing Protocol. The app:control element MUST appear only once in an Entry. The app:control element is considered foreign markup as defined in Section 6 of [RFC4287].

The app:control element and its child elements MAY be included in Atom Feed or Entry Documents.

The app:control element can contain an "app:draft" element as defined below, and it can contain extension elements as defined in Section 6 of [RFC4287].

13.1.1. The "app:draft" Element

The inclusion of the "app:draft" element represents a request by the client to control the visibility of a Member Resource. The app:draft element MAY be ignored by the server.

The number of app:draft elements in app:control MUST be zero or one. The content of an app:draft element MUST be one of "yes" or "no". If the element contains "no", this indicates a client request that the Member Resource be made publicly visible. If the app:draft element is not present, then servers that support the extension MUST behave as though an app:draft element containing "no" was sent.

14. Securing the Atom Publishing Protocol

The Atom Publishing Protocol is based on HTTP. Authentication requirements for HTTP are covered in Section 11 of [RFC2616].

The use of authentication mechanisms to prevent POSTing or editing by unknown or unauthorized clients is RECOMMENDED but not required. When authentication is not used, clients and servers are vulnerable to trivial spoofing, denial-of-service, and defacement attacks. However, in some contexts, this is an acceptable risk.

The type of authentication deployed is a local decision made by the server operator. Clients are likely to face authentication schemes that vary across server deployments. At a minimum, client and server implementations MUST be capable of being configured to use HTTP Basic Authentication [RFC2617] in conjunction with a connection made with TLS 1.0 [RFC2246] or a subsequent standards-track version of TLS (such as [RFC4346]), supporting the conventions for using HTTP over TLS described in [RFC2818].

The choice of authentication mechanism will impact interoperability. The minimum level of security referenced above (Basic Authentication with TLS) is considered good practice for Internet applications at the time of publication of this specification and sufficient for establishing a baseline for interoperability. Implementers are encouraged to investigate and use alternative mechanisms regarded as equivalently good or better at the time of deployment. It is RECOMMENDED that clients be implemented in such a way that new authentication schemes can be deployed.

Because this protocol uses HTTP response status codes as the primary means of reporting the result of a request, servers are advised to respond to unauthorized or unauthenticated requests using an appropriate 4xx HTTP response code (e.g., 401 "Unauthorized" or 403 "Forbidden") in accordance with [RFC2617].

15. Security Considerations

The Atom Publishing Protocol is based on HTTP and thus subject to the security considerations found in Section 15 of [RFC2616].

The threats listed in this section apply to many protocols that run under HTTP. The Atompub Working Group decided that the protection afforded by running authenticated HTTP under TLS (as described in Section 14) was sufficient to mitigate many of the problems presented by the attacks listed in this section.

15.1. Denial of Service

Atom Publishing Protocol server implementations need to take adequate precautions to ensure malicious clients cannot consume excessive server resources (CPU, memory, disk, etc.).

15.2. Replay Attacks

Atom Publishing Protocol server implementations are susceptible to replay attacks. Specifically, this specification does not define a means of detecting duplicate requests. Accidentally sent duplicate requests are indistinguishable from intentional and malicious replay attacks.

15.3. Spoofing Attacks

Atom Publishing Protocol implementations are susceptible to a variety of spoofing attacks. Malicious clients might send Atom Entries containing inaccurate information anywhere in the document.

15.4. Linked Resources

Atom Feed and Entry Documents can contain XML External Entities as defined in Section 4.2.2 of [REC-xml]. Atom implementations are not required to load external entities. External entities are subject to the same security concerns as any network operation and can alter the semantics of an Atom document. The same issues exist for Resources linked to by Atom elements such as atom:link and atom:content.

15.5. Digital Signatures and Encryption

Atom Entry and Feed Documents can contain XML Digital Signatures [REC-xmldsig-core] and can be encrypted using XML Encryption [REC-xmlenc-core] as specified in Section 5 of [RFC4287]. Handling of signatures and encrypted elements in Atom documents is discussed in Sections 5 and 6.3 of [RFC4287].

Neither servers nor clients are under any obligation to support encryption and digital signature of Entries or Feeds, although it is certainly possible that in some installations, clients or servers might require signing or encrypting of the documents exchanged in the Atom Protocol.

Because servers are allowed (and in some cases, expected) to modify the contents of an Entry Document before publishing it, signatures within an entry are only likely to be useful to the server to which the entry is being sent. Clients cannot assume that the signature will be valid when viewed by a third party, or even that the server will publish the client’s signature.

A server is allowed to strip client-applied signatures, to strip client-applied signatures and then re-sign with its own public key, and to oversign an entry with its own public key. The meaning to a third party of a signature applied by a server is the same as a signature from anyone, as described in [RFC4287]. It is RECOMMENDED that a server that is aware that it has changed any part of an Entry Document that was signed by the client should strip that signature before publishing the entry in order to prevent third parties from trying to interpret a signature that cannot be validated.

15.6. URIs and IRIs

Atom Publishing Protocol implementations handle URIs and IRIs. See Section 7 of [RFC3986] and Section 8 of [RFC3987] for security considerations related to their handling and use.

The Atom Publishing Protocol leaves the server in control of minting URIs. The use of any client-supplied data for creating new URIs is subject to the same concerns as described in the next section.

15.7. Code Injection and Cross Site Scripting

Atom Feed and Entry Documents can contain a broad range of content types including code that might be executable in some contexts. Malicious clients could attempt to attack servers or other clients by injecting code into a Collection Document's Entry or Media Resources.

Server implementations are strongly encouraged to verify that client-supplied content is safe prior to accepting, processing, or publishing it. In the case of HTML, experience indicates that verification based on a white list of acceptable content is more effective than a black list of forbidden content.

Additional information about XHTML and HTML content safety can be found in Section 8.1 of [RFC4287].

16. IANA Considerations

This specification uses two new media types that conform to the registry mechanism described in [RFC4288], a new message header that conforms to the registry mechanism described in [RFC3864], and two new link relations that conform to the registry mechanism described in [RFC4287].

16.1. Content-Type Registration for 'application/atomcat+xml'

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

MIME media type name: application

MIME subtype name: atomcat+xml

Required 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 RFC 5023.

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: RFC 5023.

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: .atomcat

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

Intended usage: COMMON

Author/Change controller: IETF (iesg@ietf.org) Internet Engineering Task Force

16.2. Content-Type Registration for 'application/atomsvc+xml'

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

MIME media type name: application

MIME subtype name: atomsvc+xml

Required 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 RFC 5023.

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: RFC 5023.

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: .atomsvc

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: IETF (iesg@ietf.org) Internet Engineering Task Force

16.3. Header Field Registration for 'SLUG'

Header field name: SLUG

Applicable protocol: http [RFC2616]

Status: standard.

Author/Change controller: IETF (iesg@ietf.org) Internet Engineering Task Force

Specification document(s): RFC 5023.

Related information: None.

16.4. The Link Relation Registration "edit"

Attribute Value: edit

Description: An IRI of an editable Member Entry. When appearing within an atom:entry, the href IRI can be used to retrieve, update, and delete the Resource represented by that Entry.

Expected display characteristics: Undefined; this relation can be used for background processing or to provide extended functionality without displaying its value.

Security considerations: Automated agents should take care when this relation crosses administrative domains (e.g., the URI has a different authority than the current document).

16.5. The Link Relation Registration "edit-media"

Attribute Value: edit-media

Description: An IRI of an editable Media Resource. When appearing within an atom:entry, the href IRI can be used to retrieve, update, and delete the Media Resource associated with that Entry.

Expected display characteristics: Undefined; this relation can be used for background processing or to provide extended functionality without displaying its value.

Security considerations: Automated agents should take care when this relation crosses administrative domains (e.g., the URI has a different authority than the current document).

16.6. The Atom Format Media Type Parameter

IANA has added a reference to this specification in the 'application/atom+xml' media type registration.

17. References

17.1. Normative References

[REC-xml] Yergeau, F., Paoli, J., Bray, T., Sperberg-McQueen, C., and E. Maler, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", World Wide Web Consortium Recommendation REC-xml-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml-20060816>.

[REC-xml-infoset] Cowan, J. and R. Tobin, "XML Information Set (Second Edition)", World Wide Web Consortium Recommendation REC-xml-infoset-20040204, February 2004, <http://www.w3.org/TR/2004/REC-xml-infoset-20040204>.

[REC-xml-names] Hollander, D., Bray, T., Tobin, R., and A. Layman, "Namespaces in XML 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xml-names-20060816, August 2006, <http://www.w3.org/TR/2006/REC-xml-names-20060816>.

[REC-xmlbase] Marsh, J., "XML Base", W3C REC W3C.REC-xmlbase-20010627, June 2001, <http://www.w3.org/TR/2001/REC-xmlbase-20010627>.

[REC-xmldsig-core] Solo, D., Reagle, J., and D. Eastlake, "XML-Signature Syntax and Processing", World Wide Web Consortium Recommendation REC-xmldsig-core-20020212, February 2002, <http://www.w3.org/TR/2002/REC-xmldsig-core-20020212>.

[REC-xmlenc-core] Eastlake, D. and J. Reagle, "XML Encryption Syntax and Processing", World Wide Web Consortium Recommendation REC-xmlenc-core-20021210, December 2002, <http://www.w3.org/TR/2002/REC-xmlenc-core-20021210>.

[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.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[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.

[RFC4287] Nottingham, M. and R. Sayre, "The Atom Syndication Format", RFC 4287, December 2005.

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

17.2. Informative References

[NOTE-detect-lost-update]
Nielsen, H. and D. LaLiberte, "Editing the Web: Detecting the Lost Update Problem Using Unreserved Checkout", World Wide Web Consortium NOTE NOTE-detect-lost-update, May 1999, <http://www.w3.org/1999/04/Editing/>.

[REC-webarch]
Walsh, N. and I. Jacobs, "Architecture of the World Wide Web, Volume One", W3C REC REC-webarch-20041215, December 2004, <http://www.w3.org/TR/2004/REC-webarch-20041215>.

[RNC] Clark, J., "RELAX NG Compact Syntax", December 2001, <http://www.oasis-open.org/committees/relax-ng/compact-20021121.html>.

Appendix A. Contributors

The content and concepts within are a product of the Atom community and the Atompub Working Group.

Appendix B. RELAX NG Compact Schema

This appendix is informative.

The Relax NG schema explicitly excludes elements in the Atom Protocol namespace that are not defined in this revision of the specification. Requirements for Atom Protocol processors encountering such markup are given in Sections 6.2 and 6.3 of [RFC4287].

The Schema for Service Documents:

```
# -*- rnc -*- # RELAX NG Compact Syntax Grammar for the Atom Protocol

namespace app = "http://www.w3.org/2007/app"
namespace atom = "http://www.w3.org/2005/Atom"
namespace xsd = "http://www.w3.org/2001/XMLSchema"
namespace xhtml = "http://www.w3.org/1999/xhtml"
namespace local = ""

start = appService

# common:attrs

atomURI = text

appCommonAttributes =
   attribute xml:base { atomURI }?,
   attribute xml:lang { atomLanguageTag  }?,
   attribute xml:space {"default"|"preserved"}?,
   undefinedAttribute*

atomCommonAttributes = appCommonAttributes

undefinedAttribute = attribute * - (xml:base | xml:space  | xml:lang
  | local:*) { text }

atomLanguageTag = xsd:string {
   pattern = "([A-Za-z]{1,8}(-[A-Za-z0-9]{1,8})*)?"
}
```

```
   atomDateConstruct =
       appCommonAttributes,
       xsd:dateTime

   # app:service
   appService =
      element app:service {
         appCommonAttributes,
         ( appWorkspace+
           & extensionElement* )
      }

   # app:workspace

   appWorkspace =
      element app:workspace {
         appCommonAttributes,
         ( atomTitle
           & appCollection*
           & extensionSansTitleElement* )
      }

   atomTitle = element atom:title { atomTextConstruct }

   # app:collection

   appCollection =
      element app:collection {
         appCommonAttributes,
         attribute href { atomURI  },
         ( atomTitle
           & appAccept*
           & appCategories*
           & extensionSansTitleElement* )
      }

   # app:categories

   atomCategory =
       element atom:category {
          atomCommonAttributes,
          attribute term { text },
          attribute scheme { atomURI }?,
          attribute label { text }?,
          undefinedContent
       }
```

```
   appInlineCategories =
       element app:categories {
           attribute fixed { "yes" | "no" }?,
           attribute scheme { atomURI }?,
           (atomCategory*,
           undefinedContent)
       }

   appOutOfLineCategories =
       element app:categories {
           attribute href { atomURI },
           undefinedContent
       }

   appCategories = appInlineCategories | appOutOfLineCategories

   # app:accept

   appAccept =
      element app:accept {
            appCommonAttributes,
            ( text? )
      }

   # Simple Extension

   simpleSansTitleExtensionElement =
      element * - (app:*|atom:title) {
         text
      }

   simpleExtensionElement =
      element * - (app:*) {
         text
      }

   # Structured Extension

   structuredSansTitleExtensionElement =
      element * - (app:*|atom:title) {
         (attribute * { text }+,
            (text|anyElement)*)
       | (attribute * { text }*,
          (text?, anyElement+, (text|anyElement)*))
      }
```

```
   structuredExtensionElement =
      element * - (app:*) {
         (attribute * { text }+,
            (text|anyElement)*)
       | (attribute * { text }*,
          (text?, anyElement+, (text|anyElement)*))
      }

   # Other Extensibility

   extensionSansTitleElement =
    simpleSansTitleExtensionElement|structuredSansTitleExtensionElement

   extensionElement = simpleExtensionElement |
      structuredExtensionElement

   undefinedContent = (text|anyForeignElement)*

   # Extensions

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

   anyForeignElement =
       element * - app:* {
          (attribute * { text }
           | text
           | anyElement)*
       }

   atomPlainTextConstruct =
       atomCommonAttributes,
       attribute type { "text" | "html" }?,
       text

   atomXHTMLTextConstruct =
       atomCommonAttributes,
       attribute type { "xhtml" },
       xhtmlDiv

   atomTextConstruct = atomPlainTextConstruct | atomXHTMLTextConstruct
```

```
   anyXHTML = element xhtml:* {
       (attribute * { text }
        | text
        | anyXHTML)*
   }

   xhtmlDiv = element xhtml:div {
     (attribute * { text }
      | text
      | anyXHTML)*
   }

   # EOF
```

The Schema for Category Documents:

```
   # -*- rnc -*- # RELAX NG Compact Syntax Grammar for the Atom Protocol

   namespace app = "http://www.w3.org/2007/app"
   namespace atom = "http://www.w3.org/2005/Atom"
   namespace xsd = "http://www.w3.org/2001/XMLSchema"
   namespace local = ""

   start = appCategories

   atomCommonAttributes =
      attribute xml:base { atomURI }?,
      attribute xml:lang { atomLanguageTag }?,
      undefinedAttribute*

   undefinedAttribute = attribute * - (xml:base | xml:lang | local:*) {
     text }

   atomURI = text

   atomLanguageTag = xsd:string {
      pattern = "([A-Za-z]{1,8}(-[A-Za-z0-9]{1,8})*)?"
   }

   atomCategory =
       element atom:category {
          atomCommonAttributes,
          attribute term { text },
          attribute scheme { atomURI }?,
          attribute label { text }?,
          undefinedContent
       }
```

```
   appInlineCategories =
       element app:categories {
           attribute fixed { "yes" | "no" }?,
           attribute scheme { atomURI }?,
           (atomCategory*,
           undefinedContent)
       }

   appOutOfLineCategories =
       element app:categories {
           attribute href { atomURI },
           (empty)
       }

   appCategories = appInlineCategories | appOutOfLineCategories

   # Extensibility

   undefinedContent = (text|anyForeignElement)*

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

   anyForeignElement =
       element * - atom:* {
          (attribute * { text }
           | text
           | anyElement)*
       }

   # EOF
```

Authors' Addresses

Joe Gregorio (editor)
Google

EMail: joe@bitworking.org
URI: http://bitworking.org/

Bill de hOra (editor)
NewBay Software

EMail: bill@dehora.net
URI: http://dehora.net/

Full Copyright Statement

Intellectual Property