

# The Atom API

Mark Pilgrim

ApacheCon 2003

Backchannel: `#apollo3` on `irc.freenode.net`

# Evolution of weblogging APIs

- LiveJournal API
- Manila API
- Blogger API
- Metaweblog API
- Atom API

# LiveJournal API

- GPL weblog/community system
- API started as “flat” HTTP-based protocol
- Other variants were added later
- <http://livejournal.com/doc/server/ljp.csp.flat.protocol.html>

# LiveJournal API

- All functions use HTTP POST
- Arguments are form-encoded
- Values returned in plain text
  - key-value pairs
  - separated by carriage returns

# LiveJournal API request

POST /interface/flat HTTP/1.1

Host: www.livejournal.com

Content-type: application/x-www-form-urlencoded

mode=login&user=test&password=test

# LiveJournal API response

```
HTTP/1.1 200 OK  
Content-Type: text/plain
```

```
name  
Mr. Test Account  
success  
OK  
message  
Hello Test Account!
```

# LiveJournal API: problems

- Escaping issues with request
- Escaping issues with response
- Unmarshalling the output
- What's a carriage return?
- Never implemented outside LiveJournal
- Passwords sent in the clear, as plain text

# Manila API

- CMS developed by UserLand Software
- Comprehensive API, supports everything you can do through Manila's web interface
- XML-RPC and RPC-style SOAP
- <http://xmlrpc.com/manilaRpc>

# Manila API: problems

- RPC-style interface not easily extensible
- Interface specific to Manila (by design)
- Never adopted outside Userland
- Passwords sent in the clear, as plain text

# Blogger API

- Weblog publishing tool by Pyra Labs
  - now owned by Google
- XML-RPC only
- Adopted by UserLand, and later by SixApart and many others
- Currently the most widely adopted weblogging web service
- [http://www.blogger.com/developers/api/1\\_docs/](http://www.blogger.com/developers/api/1_docs/)

# Blogger API request

```
POST /api/RPC2 HTTP/1.1  
Host: plant.blogger.com  
Content-Type: text/xml
```

```
<?xml version='1.0'?>  
<methodCall>  
  <methodName>blogger.newPost</methodName>  
  <params>  
    <param>  
      <value>  
        <string>APP_KEY</string>  
      </value>  
    </param>
```

# Blogger API request

```
<param>
  <value>
    <string>BLOG_ID</string>
  </value>
</param>
<param>
  <value>
    <string>USERNAME</string>
  </value>
</param>
<param>
  <value>
    <string>PASSWORD</string>
  </value>
</param>
```

# Blogger API request

```
<param>
  <value>
    <string>ENTRY_TEXT</string>
  </value>
</param>
<param>
  <value>
    <boolean>PUBLISH</boolean>
  </value>
</param>
<params>
</methodCall>
```

# Blogger API problems

- Limited functionality
  - `newPost`, `editPost`, `getUsersBlogs`,  
`getUserInfo`, `getTemplate`, `setTemplate`
- Interface is Blogger-centric
  - No titles
  - No dates
  - No flags beyond “draft”/“publish”
- No extensibility mechanism
- Passwords are sent in the clear, as plain text

# XML-RPC problems

- Historical quirks
  - Timezones?
  - Unicode?
  - https?
- RPC-style makes extensibility difficult
- Verbose serialization makes debugging difficult
- Not designed to integrate with other XML technologies

# Metaweblog API

- Developed by UserLand in response to perceived weaknesses of Blogger API
- "Solves" extensibility problem by using structs for all inputs and outputs
- "Solves" functionality problem by defining new methods and documenting new parameters (within The Struct) for existing methods
- Supported by some other vendors
- <http://www.xmlrpc.com/metaWeblogApi>

# The Struct

From the Metaweblog API spec:

The MetaWeblog API uses an XML-RPC struct to represent a weblog post. Rather than invent a new vocabulary for the metadata of a weblog post, we use the vocabulary for an item in RSS 2.0. So you can refer to a post's title, link and description; or its author, comments, enclosure, guid, etc. using the already-familiar names given to those elements in RSS 2.0.

# The RSS 2.0 “data model”

```
<item>
  <title>My Weblog Entry</title>
  <description>First post!</description>
  <pubDate>
    Mon, 13 Oct 2003 13:29:54 GMT
  </pubDate>
  <author>Mark Pilgrim</author>
  <category>Unfiled</category>
</item>
```

# Creating “The Struct”

```
>> import xmlrpclib
>>> server = xmlrpclib.ServerProxy(
    'http://www.example.com/RPC2')
>>> server.metaWeblog.newPost(
    BLOG_ID, USERNAME, PASSWORD,
    {'title': 'My Weblog Entry',
     'description': 'First post!',
     'dateCreated': '2003-10-13T13:29:54',
     'author': 'Mark Pilgrim',
     'category': 'Unfiled'},
    xmlrpclib.True)
```

# Metaweblog API: request

```
POST /RPC2 HTTP/1.0  
Host: www.example.com  
Content-Type: text/xml
```

```
<?xml version='1.0'?>  
<methodCall>  
  <methodName>  
    metaWeblog.newPost  
  </methodName>
```

# Metaweblog API: request

```
<params>
  <param>
    <value>
      <string>
        BLOG_ID
      </string>
    </value>
  </param>
```

# Metaweblog API: request

```
<param>  
  <value>  
    <string>  
      USERNAME  
    </string>  
  </value>  
</param>
```

# Metaweblog API: request

```
<param>  
  <value>  
    <string>  
      PASSWORD  
    </string>  
  </value>  
</param>
```

# Metaweblog API: request

```
<param>
  <value>
    <struct>
      <member>
        <name>
          title
        </name>
        <value>
          <string>
            My Weblog Entry
          </string>
        </value>
      </member>
```

# Metaweblog API: request

```
<member>  
  <name>  
    description  
  </name>  
  <value>  
    <string>  
      First post!  
    </string>  
  </value>  
</member>
```

# Metaweblog API: request

```
<member>  
  <name>  
    author  
  </name>  
  <value>  
    <string>  
      Mark Pilgrim  
    </string>  
  </value>  
</member>
```

# Metaweblog API: request

```
<member>  
  <name>  
    category  
  </name>  
  <value>  
    <string>  
      unfiled  
    </string>  
  </value>  
</member>
```

# Metaweblog API: request

```
<member>
  <name>
    dateCreated
  </name>
  <value>
    <dateTime.iso8601>
      2003-10-13T13:29:54
    </dateTime.iso8601>
  </value>
</member>
```

# Metaweblog API: request

```
        </struct>
    </value>
</param>
<param>
    <value>
        <boolean>1</boolean>
    </value>
</param>
</params>
</methodCall>
```

# Metaweblog API: problems

- Despite the spec's claim that the vocabulary "comes from RSS 2.0", it doesn't really
- Element names don't match
  - <pubDate>
  - <dateCreated>
- Date formats don't match
  - RFC-822
  - ISO-8601
- XML-RPC dates have no concept of time zones
- Other problems inherent in XML-RPC

# The category problem

- “The Struct” uses element names as unique keys
- But entries can have multiple categories
- Spec dodges this issue by defining a separate <categories> element within The Struct which is an array of strings
- Other cases are simply impossible
  - Multiple authors

# The category problem

- RSS categories can have attributes too
  - value is the name of the category
  - domain attribute specifies the domain in which the category name resides.

"If an element has both attributes and a value, make the element a struct, include the attributes as sub-elements, and create a sub-element for the value with the name `_value`. Note that this means that no element can be passed through the API that has an attribute whose name is `_value`."

# The attribute problem

- Some elements in RSS (`source`, `enclosure`, and `category`) can have attributes
- For `enclosure`, the MetaWeblog API tells us to "pass a struct with sub-elements whose names match the names of the attributes according to the RSS 2.0 spec, `url`, `length` and `type`"
- For `source`, "pass a struct with sub-elements, `url` and `name`"
- Multiple categories with domains are impossible

# The extensibility problem

- RSS 2.0 is extensible through namespaces
- RSS 2.0 is the data model for the Metaweblog API
- Therefore, the Metaweblog API should be extensible through namespaces

(This syllogism has been brought to you by Clay Shirky)

# The extensibility problem

- But XML-RPC doesn't support namespaces

Sorry, Clay

# The extensibility “solution”

- The Metaweblog spec says:
  - “If you wish to transmit an element that is part of a namespace, include a sub-struct in the struct passed to `newPost` and `editPost` whose name is the URL that specifies the namespace. The sub-element(s) of the struct are the value(s) from the namespace that you wish to transmit.”
- No one actually does this
  - `mt_allow_comments`
  - `mt_allow_pings`

# And did I mention...

- Passwords are sent in the clear, as plain text

# Metaweblog API: summary

In case you lost track, what we have here is an RPC-based API that starts with an XML-centric data model (RSS 2.0), treats it as a series of simple key-value pairs, shoves all those pairs into a struct, defines separate special cases for everything that doesn't fit, ignores everything that isn't handled by the special cases, reinvents the concept of XML namespaces, and then serializes it all in a verbose XML format.

# Metaweblog API: summary

So we've reinvented XML, over RPC,  
over XML. Badly.

And passwords are still sent in the clear,  
as plain text.

# Atom API: goals

- 100% vendor neutral
- Implemented by everybody
- Implementable by everybody
  - hosted accounts
  - CGI only
  - no .htaccess
- Freely and easily extensible by anybody
- Cleanly and thoroughly specified
- Secure

# Atom API: architecture

- Doc-literal (not RPC)
- Small, well-defined vocabulary for specific purpose
  - Not trying to be all things to all people
  - not trying to reinvent WebDAV
- Take full advantage of XML
- Take full advantage of HTTP
- No cleartext passwords

# API discovery

- Previous APIs had no standard for API discovery
- Left it up to the end user to provide exact API URL
  - <http://example.com/mt/mt-xmlrpc.cgi>
- RSD
- Some servers implemented undocumented functions
  - deletePost
- Client software had to guess

# Atom API discovery

- Only assumes user knows their home page
- LINK tag in HEAD of home page points to Atom introspection file
- Atom introspection file lists supported supported functions and extensions

# Atom introspection file

- Lists supported functions and extensions
- Simple, well-defined XML format
- Vendors can extend introspection file with XML namespaces

# Atom auto-discovery

```
<html>
<head>
<title>My Weblog</title>
<link rel="service.edit"
      type="application/x.atom+xml"
      href="/myblog/atom.cgi/introspection"
      title="Atom API">
</head>
<body>
...
```

# Atom introspection file

```
<?xml version="1.0" encoding="utf-8"?>
<introspection xmlns="http://purl.org/atom/ns#">
  <search-entries>
    http://example.com/myblog/atom.cgi/search
  </search-entries>
  <create-entry>
    http://example.com/myblog/atom.cgi/edit
  </create-entry>
  <edit-template>
    http://example.com/atom.cgi/templates
  </edit-template>
  <user-prefs>
    http://example.com/myblog/atom.cgi/prefs
  </user-prefs>
  <categories>
    http://example.com/atom.cgi/categories
  </categories>
</introspection>
```

# Finding entries

- Use search-entries URI specified in introspection file
- Add query string parameters such as atom-last (recent entries)
- More complex examples defined in Atom API spec
  - atom-all
  - date ranges

# Listing recent entries

```
GET /myblog/atom.cgi/search?atom-last=20 HTTP/1.1
Host: example.com
HTTP/1.1 200 OK
Content-Type: application/x.atom+xml
```

```
<search-results xmlns="http://purl.org/atom/ns#">
  <entry>
    <title>My Second Post</title>
    <id>http://example.com/atom.cgi/edit/2</id>
  </entry>
  <entry>
    <title>My First Post</title>
    <id>http://example.com/atom.cgi/edit/1</id>
  </entry>
</search-results>
```

# The Edit URI

- Address for editing a specific entry
- Returned in the list of entries
- Also returned after creating a new entry
- <http://example.com/atom.cgi/edit/1>
- Naming scheme is entirely up to the server

# Dispatching methods

- LiveJournal uses mode= parameter
- Manila API, Blogger API, Metaweblog API uses function name within XML-RPC body
- SOAP uses SOAPAction header and function name within SOAP body
- Atom API uses edit URI + HTTP verb

# Atom dispatching methods

- Edit URI + GET = retrieve entry
  - Edit URI + PUT = modify entry
  - Edit URI + DELETE = delete entry
  - Create URI + POST = new entry
- 
- This is how HTTP is supposed to work

# Retrieving an entry

```
GET /myblog/atom.cgi/edit/1 HTTP/1.1  
Host: example.com
```

# Retrieving an entry: response

```
HTTP/1.1 200 OK
```

```
Content-Type: application/x.atom+xml
```

```
<?xml version="1.0" encoding="utf-8"?>  
<entry xmlns="http://purl.org/atom/ns#">  
  <title>My First Entry</title>  
  <summary>My First Entry Excerpt (generally  
  plaintext)</summary>  
  <author>  
    <name>Bob B. Bobbington</name>  
    <email>bob@example.com</email>  
    <url>http://homepage.example.com/</url>  
  </author>
```

# Retrieving an entry: response

```
<issued>2003-10-15T02:29:29</issued>
<created>2003-10-15T04:10:58Z</created>
<modified>2003-10-15T04:22:03Z</modified>
<link>
http://example.com/myblog/archives/2003/11/19/My
  First\_Entry.html
</link>
<id>urn:example-com:myblog:1</id>
<content type="application/xhtml+xml"
  xml:lang="en">
  <div xmlns="http://www.w3.org/1999/xhtml">
    <p>Hello, <em>weblog</em> world!</p>
  </div>
</content>
</entry>
```

# Atom content model

- `type` attribute specifies MIME type
- `xml:lang` specifies language
- XHTML can be included inline
  - In appropriate namespace
- Legacy HTML can be included
  - Escaped or in CDATA block
  - `mode="escaped"` on content element
- Binary data can be included
  - Base64 encoded
  - `mode="base64"` on content element

# Creating a new entry

```
POST /myblog/atom.cgi/edit HTTP/1.1
Host: example.com
Content-Type: application/x.atom+xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://purl.org/atom/ns#">
  <title>My Entry Title</title>
  <created>2003-11-17T12:29:29Z</created>
  <content type="application/xhtml+xml"
    xml:lang="en">
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>Hello, <em>weblog</em> world!</p>
      <p>This is my third post</p></div>
    </content>
  </entry>
```

# Creating a new entry: response

HTTP/1.1 201 Created

Location: <http://example.com/myblog/atom.cgi/edit/3>

# Full use of HTTP

- Proper HTTP status codes
  - New entry generates HTTP 201
  - Modify entry generates HTTP 205
  - Authorization generates HTTP 401
  - Authentication failure generates HTTP 403
- Location header specifies edit URI of newly created resource
- Allow full use of Etags, Last-Modified headers, gzip compression
- This is how HTTP is supposed to work

# Extensibility thru namespaces

```
POST /myblog/atom.cgi HTTP/1.1
Host: example.com
Content-Type: application/x.atom+xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<entry
  xmlns="http://purl.org/atom/ns#"
  xmlns:mt="http://www.movabletype.org/atom/ns#">
  <title>My Entry Title</title>
  <created>2003-11-17T12:29:29Z</created>
  <mt:allowComments>1</mt:allowComments>
  <content type="application/xhtml+xml" xml:lang="en">
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>Hello, <em>weblog</em> world!</p>
    </div>
  </content>
</entry>
```

# Modifying an existing entry

```
PUT /myblog/atom.cgi/edit/1 HTTP/1.1
Host: example.com
Content-Type: application/x.atom+xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://purl.org/atom/ns#">
  <title>My First Entry</title>
  <summary>My First Entry Excerpt (generally
  plaintext)</summary>
  <author>
    <name>Bob B. Bobbington</name>
    <email>bob@example.com</email>
    <url>http://homepage.example.com/</url>
  </author>
```

# Modifying an existing entry

```
<issued>2003-10-15T02:29:29</issued>
<created>2003-10-15T04:10:58Z</created>
<modified>2003-10-15T04:22:03Z</modified>

<link>http://example.com/myblog/archives/2003/
11/19/My_First_Entry.html</link>
<id>urn:example-com:myblog:1</id>
<content type="application/xhtml+xml"
xml:lang="en">
  <div xmlns="http://www.w3.org/1999/xhtml">
    <p>Hello, <em>entire</em> world!</p></div>
  </content>
</entry>
```

# Modifying an existing entry: response

HTTP/1.1 205 Reset Content

# Deleting an entry

```
DELETE /myblog/atom.cgi/edit/3 HTTP/1.1  
Host: example.com
```

# Deleting an entry: response

HTTP/1.1 200 OK

# Other uses for the Atom API

- Posting comments
- Managing users
- Managing user preferences
- Managing site templates
- Managing categories
- <http://bitworking.org/rfc/draft-gregorio-07.html>

# Why Atom isn't SOAP

- **Atom format is already an XML wrapper around entry content**
- **Zero vendor interest in anything stronger or more exotic than HTTP authentication**
- **Zero vendor interest in transport independence**

# Why Atom isn't RDF

- **A vocal minority would like every XML format to be RDF**
- **RDF does not solve any of the design goals**
  - **Adds a lot of complexity though**

# Why Atom isn't WebDAV

- **Atom is designed for editing episodic web sites**
  - Other uses are fine too
- **Atom provides mediated access to multiple vendor-specific backends via a common API**
- **Atom must be implementable by independent site maintainers on hosted accounts (CGI only, no .htaccess)**

# Further reading

- <http://bitworking.org/rfc/draft-gregorio-07.html>
- <http://intertwingly.net/wiki/pie/>
- <http://www.imc.org/atom-syntax/>
- [\*\*http://xml.com/pub/a/2003/10/15/dive.html\*\*](http://xml.com/pub/a/2003/10/15/dive.html)
- [\*\*http://diveintomark.org/public/2003/11/atomapi.pdf\*\*](http://diveintomark.org/public/2003/11/atomapi.pdf)

# The Atom API

Mark Pilgrim

ApacheCon 2003