

**CONTENTS INCLUDE:**

- Evolution of RSS Standards
- RSS 2.0 Feed Elements
- Atom 1.0 Feed Elements
- The Metaweblog API
- The Blogger API
- The Atom Protocol
- Hot Tips and more...

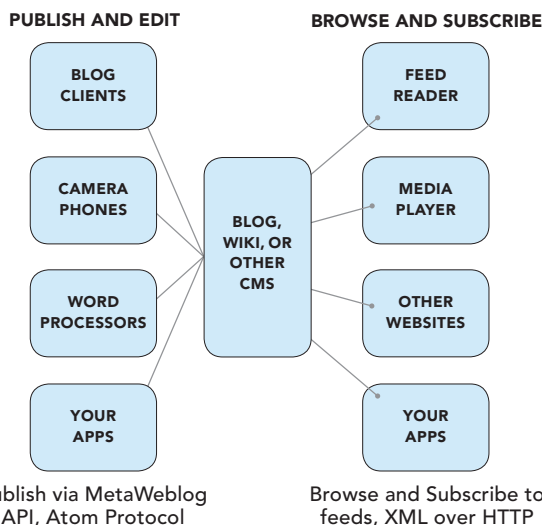
# RSS and Atom

By Dave Johnson

## PUBLISH AND SUBSCRIBE WITH RSS AND ATOM

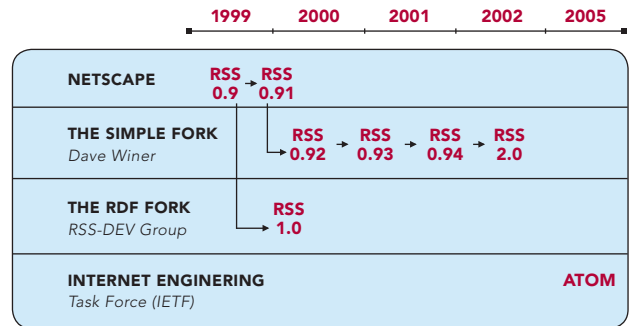
The explosive growth of RSS and Atom feeds on the internet make it easier than ever before for your software to publish, edit, monitor and extract data from the web. That's why feeds are the core of a host of new RESTful web services from simple blog publishing protocols to Google's expansive GData and OpenSocial APIs. You'll find this reference card useful whether you are creating and serving, or subscribing to and parsing feeds. It lists the XML elements in the most widely used feed formats and it illustrates the relationship between multiple variants of RSS and Atom. We list and explain the methods in the XML-RPC based Blogger and MetaWeblog API. And, we provide a guide to the new RESTbased Atom web publishing protocol.

RSS and Atom make it easy to read and write the web. Applications can use the Atom Publishing Protocol (RFC-5023) and the MetaWeblog API to publish any type of content to blog, wiki and CMS servers. And servers can make any type of content available to media players, feed reader and other applications via RSS and Atom formats.



## EVOLUTION OF RSS STANDARDS

Depending on whom you ask, RSS stands for RDF Site Summary, Rich Site Summary, Really Simple Syndication or just RSS. The diagram below shows the evolution of RSS and Atom feed formats. There are two variants of RSS: Dave Winer's simple fork and the RSS-DEV group's RDF fork. Atom (RFC-4287) is the new standard feed format.



You can find the specifications for all of these formats online at the following locations:

**RSS 0.90**

<http://www.purplepages.ie/RSS/netscape/rss0.90.html>

**RSS 0.91 (Netscape)**

<http://www.rssboard.org/rss-0-9-1-netscape>

**RSS 0.91 (UserLand)**

<http://backend.userland.com/rss091>

**RSS 0.92**

<http://backend.userland.com/rss092>

**RSS 0.93**

<http://backend.userland.com/rss093>

**RSS 0.94**

No longer available online

**RSS 1.0**

<http://web.resource.org/rss/1.0/spec>

**RSS 2.0**

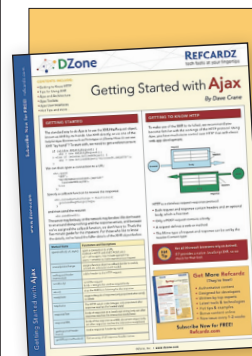
No longer available online

**RSS 2.0.1**

<http://blogs.law.harvard.edu/tech/rss>

**Atom 1.0**

<http://www.atomenabled.org/developers/syndication/atom-format-spec.php>



## Get More Refcardz (They're free!)

- Authoritative content
- Designed for developers
- Written by top experts
- Latest tools & technologies
- Hot tips & examples
- Bonus content online
- New issue every 1-2 weeks

**Subscribe Now for FREE!**  
**Refcardz.com**

## RSS 2.0 FEED ELEMENTS

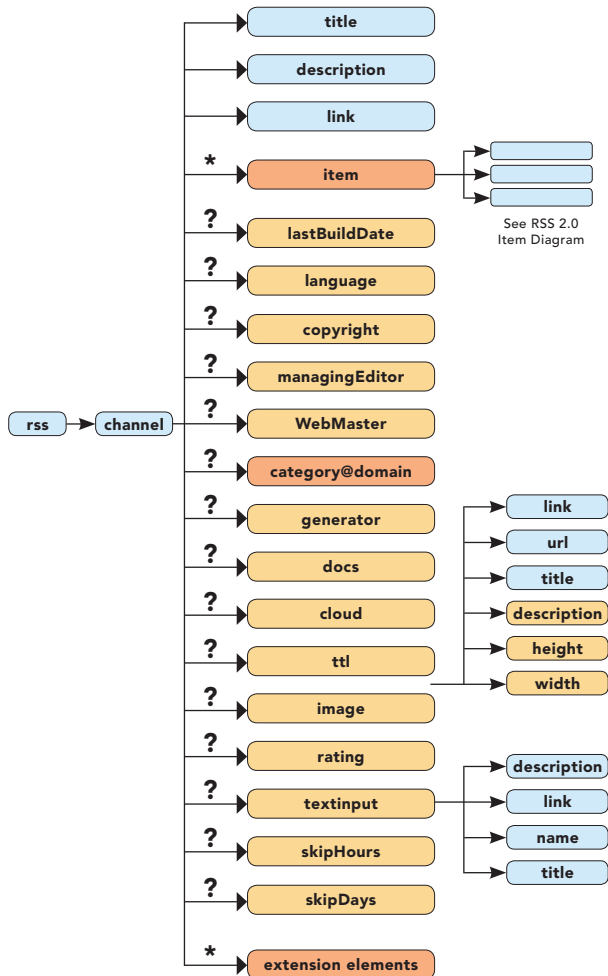
The root element is `<rss>`, it contains one `<channel>` element, which in turn contains `<item>` elements. Dates are represented in RFC-822 format. The RSS 2.0 diagram below is broken into two parts; first we show the feed level metadata under `channel` – item element children are omitted.

### Feed Diagram Key

The feed diagrams use the following notations to indicate required elements, cardinality, containment and XML attributes.

- Required XML element
- ? □ Zero or one
- \* □ Zero or more
- Containment
- @ XML element attribute

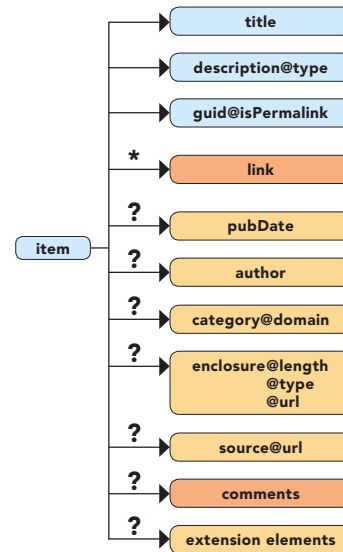
### Feed Diagram



You can extend RSS by adding your own extension elements, i.e. new XML elements, as long as they are placed in their own XML namespace.

### Feed Diagram

The second part of the RSS 2.0 diagram shows the item element and its children. Item content is carried in the `<description>` element and is represented as escaped HTML.



### RSS 2.0 Feed Examples

Example of an RSS 2.0 feed with one item and a podcast.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0">
<channel>
<title>Example Blog</title>
<link>http://example.com/blog</link>
<item>
<title>Here is an item with a podcast</title>
<description>
My first &lt;b>podcast</b>.
</description>
<pubDate>Wed, 20 Apr 2005 13:30:45 EDT</pubDate>
<link>http://example.com/blog/20050420?id=132</link>
<enclosure url="http://example.com/casts/file1.mpg"
type="audio/mpeg3" length="13456170"/>
</item>
</channel>
</rss>
```

RSS 2.0 feeds in the wild often use extension elements instead of the standard elements of RSS. For example, this feed uses the Dublin Core `<dc:date>` and `<dc:creator>` instead of the standard `<pubDate>` and `<author>` elements.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<rss version="2.0"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
<title>Example Blog</title>
<link>http://example.com/blog</link>
<item>
<title>Here is an item that uses Dublin Core</title>
<description>Just another &lt;b>blog entry</b>.
</description>
<link>http://example.com/blog/20050420?id=133</link>
<dc:date>2005-04-20T17:41:04+5:00</dc:date>
<dc:creator>Dave Johnson</dc:creator>
</item>
</rss>
```



**Use Feed Autodiscovery to Advertise Your Feeds** If your web application or site provides feeds, advertise those feeds by listing each with an HTML `<link>` element in the HTML `<head>` of your web page. For each feed, you can specify a content-type, title and an href—as shown here:

```
<link rel="alternate"
type="application/rss+xml" title="My RSS feed"
href="http://localhost/feed.rss" />
```

## ATOM 1.0 FEED ELEMENTS

The root element of an Atom feed is the **<feed>** element, which contains metadata and a collection of **<entry>** elements.

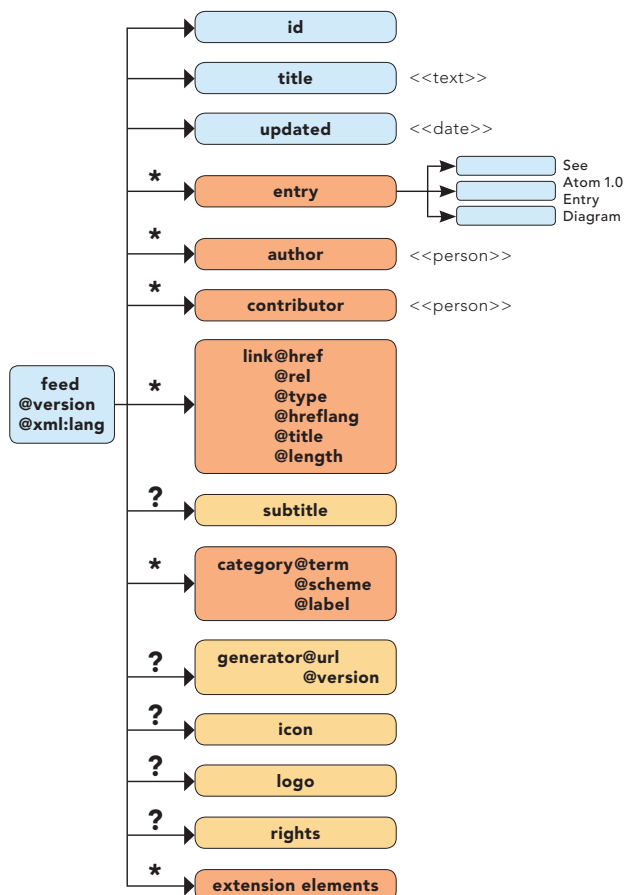
- ID must be a valid URN
- There must be a self-link containing the URI location of the feed
- Dates in Atom are represented in W3C DateTime format
- Text constructs (indicated with **<<text>>** in the diagram) may contain a type attribute with a value of text for plain text, html for escaped HTML, or xhtml for XHTML. If not present, content is assumed to be text.
- An author must be specified at the **<feed>** level or in each **<entry>**

### Feed Diagram Key

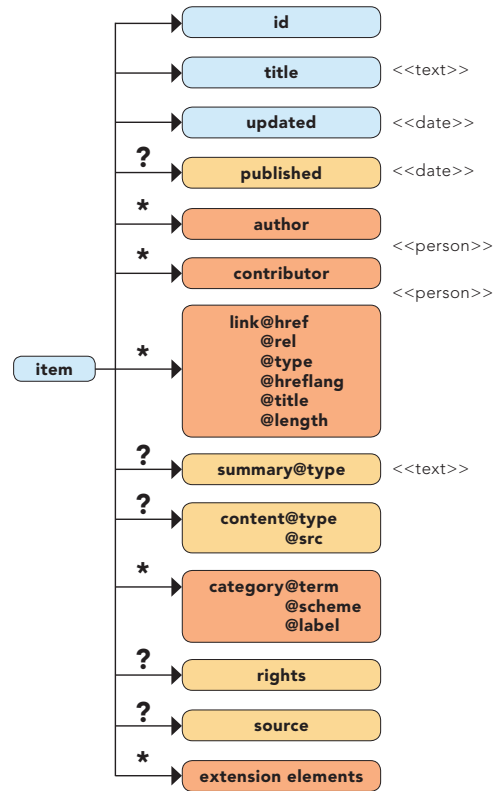
The feed diagrams use the following notations to indicate required elements, cardinality, containment and XML attributes.

- Required XML element
- ? □ Zero or one
- \* □ Zero or more
- Containment
- @ XML element attribute

### Feed Diagram



Here are the elements at the Atom **<entry>** level. Note that the **<content>** element has a type attribute like that in text construct, but it can also be set to any MIME content-type, thus allowing an Atom entry to carry any type of data.



### Atom 1.0 Example

Example Atom 1.0 feed with XHTML content.

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns='http://purl.org/atom/ns#' xml:lang='en-us'>
  <title>Oh no, Mr. Bill</title>
  <link href='http://nbc.com/sluggo/' />
  <link rel='self' href='http://nbc.com/sluggo/index.atom' />
  <updated>2005-04-06T20:25:05-08:00</updated>
  <author><name>Mr. Bill</name></author>
  <entry>
    <title>A post about stuff</title>
    <link href='http://nbc.com/sluggo/20050420?id=321' />
    <id>http://nbc.com/sluggo/20050420?id=321</id>
    <updated>2005-04-06T20:25:05-08:00</updated>
    <content type='xhtml'>
      <div xmlns='http://www.w3.org/1999/xhtml'>
        <!-- xhtml content -->
      </div>
    </content>
  </entry>
</feed>
```



### Validate your Feeds with [feedvalidator.org](http://feedvalidator.org)

If your web application or site provides feeds, validate those feeds by using the free feed validation service at [feedvalidator.org](http://feedvalidator.org). If you're serving private or behind-the-firewall feeds, then you can download the [Python] source code for the Feed Validator and run it on your own machine.

## Parsing Feeds with Java and Rome

Here's a simple example that shows how to parse and print a feed using Java and the ROME feed parser library. The example uses the `SyndFeedInput` class, which parses any format of RSS (0.9X, 1.0, 2.0) or Atom (0.3 or 1.0) to a `SyndFeed` object containing a collection of `SyndEntry` objects.

```
SyndFeedInput input = new SyndFeedInput();
SyndFeed feed = input.build(
    new InputStreamReader(inputStream));

Iterator entries =
    feed.getEntries().iterator();
while (entries.hasNext()) {
    SyndEntry entry = entries.next();
    System.out.println("Title: " + entry.getTitle());
    System.out.println("Link: " + entry.getLink());
    System.out.println("Date: " + entry.getPublishedDate());
    System.out.println("Desc: " + entry.getDescription());
    System.out.println("\n");
}
```

ROME is covered in detail in *RSS and Atom in Action*, Chapter 7. For more information on ROME, visit the project's web site at <http://rome.dev.java.net>.

## Parsing Feeds with C# & Windows RSS

Here's an example that shows how to parse and print a feed using C# and the Windows RSS Platform's Feeds API.

```
IFeedsManager fm = new FeedsManagerClass();
IFeed feed = null;
if (!fm.IsSubscribed(url)) {
    IFeedFolder rootFolder =
        (IFeedFolder)fm.RootFolder;
    feed = (IFeed)rootFolder.CreateFeed(url, url);
} else {
    feed = (IFeed)fm.GetFeedByUrl(url);
}
feed.Download();

foreach (IFeedItem item in (IFeedsEnum)feed.Items) {
    Console.Out.WriteLine("item.Title: " + item.Title);
    Console.Out.WriteLine("item.pubDate:" + item.PubDate);
    Console.Out.WriteLine("item.Desc: " + item.Description);
}
```

## THE METAWEBLOG API

The `MetaWeblog` API was created by Dave Winer; it extends the `Blogger` API by adding six new methods to allow posting and editing blog entries with better metadata than the `Blogger` API and to allow uploading of media files (image, video, etc.).

### List of Methods

Method Name	Parameters and Descriptions
<code>metaWeblog.newPost</code>	<b>string blogid, string username, string password, struct post, boolean publish</b> Creates a new post in the blog specified by <code>blogid</code> using the data from the post structure. The names in the post structure correspond to the names of the XML elements in an RSS <item>. Returns the string ID of the newly created post.
<code>metaWeblog.editPost</code>	<b>string postid, string username, string password, struct post, boolean publish</b> Updates the post specified by <code>postid</code> using data from the post structure.

### List of Methods, continued

Method Name	Parameters and Descriptions
<code>metaWeblog.getPost</code>	<b>string postid, string username, string password</b> Returns the post specified by <code>postid</code> as a post structure.
<code>metaWeblog.getRecentPosts</code>	<b>string blogid, string username, string password, int numPosts</b> Returns the most recent blog post as an array of post structures. Maximum number of posts to return is <code>numPosts</code> .
<code>metaWeblog.newMediaObject</code>	<b>string blogid, string username, string password, struct object</b> Uploads an image, video, or audio file to the blog specified by <code>blogid</code> . The file is specified by the object structure with fields <code>name</code> , <code>type</code> , and <code>bits</code> . The <code>bits</code> field is the file data encoded as Base64 data. Returns a string, which is the URL of the uploaded file.
<code>MetaWeblog.getCategories</code>	<b>string blogid, string username, string password</b> Returns the categories available in the blog specified by <code>blogid</code> as a structure of structures, each structure representing a category and having members <code>description</code> , <code>htmlUrl</code> , and <code>rssUrl</code> .

## THE BLOGGER API

The `Blogger` API was created in 2001 for `Blogger.com` and it's being replaced by the `Atom` protocol, but it is still an important API because it is the foundation of the widely used `MetaWeblog` API.

### List of Methods

Method Name	Parameters and Descriptions
<code>blogger.newPost</code>	<b>string appkey, string blogid, string username, string password, string content, boolean publish</b> Create a new blog post in the blog specified by <code>blogid</code> and <code>content</code> specified by <code>content</code> . Some servers interpret <code>publish=true</code> to mean publish publicly and <code>publish=false</code> to mean save as a private draft. Others interpret it to mean simply publish immediately. Returns a string, which is the <code>postid</code> of the new post.
<code>blogger.editPost</code>	<b>string appkey, string postid, string username, string password, string content, boolean publish</b> Update the blog post specified by <code>postid</code> with new content.
<code>blogger.deletePost</code>	<b>string appkey, string postid, string username, string password, boolean publish</b> Delete the blog post specified by <code>blogid</code> and optionally republish the blog.
<code>blogger.getRecentPosts</code>	<b>string appkey, string blogid, string username, string password, int numPosts</b> Get the most recent blog posts as an array of structures, each having members <code>dateCreated</code> , <code>userid</code> , <code>postid</code> , and <code>content</code> . Maximum number of posts to return is <code>numPosts</code> .
<code>blogger.getUsersBlogs</code>	<b>string appkey, string username, string password</b> Get the specified user's blogs as an array of structures, each having members <code>url</code> , <code>blogid</code> , and <code>blogName</code> .
<code>blogger.getUserInfo</code>	<b>string appkey, string username, string password</b> Get the specified user's information as a structure with members <code>nickname</code> , <code>userid</code> , <code>url</code> , <code>email</code> , <code>lastname</code> , <code>firstname</code> .
<code>blogger.getTemplate</code>	<b>string appkey, string blogid, string username, string password, string type</b> Get blog's template of the specified type.
<code>blogger.setTemplate</code>	<b>string appkey, string blogid, string username, string password, string template, string type</b> Change the blog's template of the specified type. The format of blog templates varies depending on the blog server.

## Making a Post with MetaWeblog API

Here is an example that uses Apache XML-RPC to post a blog entry with a title and description to a blog with a blogid, username and password. The blog server has an endpointURL.

```
import java.util.*;
import java.io.*;
import org.apache.xmlrpc.XmlRpcClient;

Hashtable post = new Hashtable();
post.put("dateCreated", new Date());
if (title != null) post.put("title", title);
post.put("description", description);

Vector params = new Vector();
params.addElement(blogid);
params.addElement(username);
params.addElement(password);
params.addElement(post);
params.addElement(Boolean.TRUE);
XmlRpcClient xmlrpc = new XmlRpcClient(endpointURL);
String newEntryId =
    (String)xmlrpc.execute("metaWeblog.newPost", params);
```

## THE ATOM PROTOCOL

Atom protocol (RFC-5023) is a REST-based protocol for creating, retrieving, updating and deleting collections of objects on a server. Objects are represented as Atom entries and collections as Atom feeds.

### Service Document

To find out what workspaces and collections are available on an Atom server, send an authenticated HTTP GET request to the server's end-point URI.

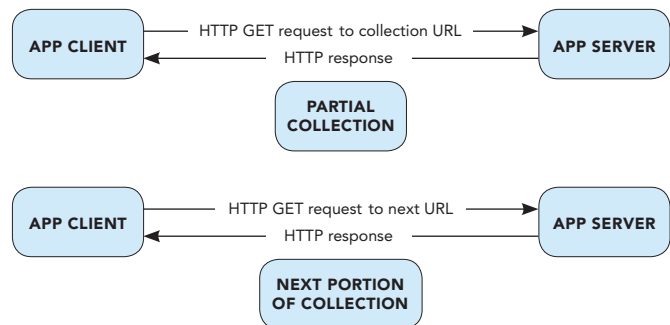


You'll get back an Atom service document like the one below, which includes one workspace that contains two collections: one of entries and one of images. Note that each collection has a collection URI.

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://www.w3.org/2007/app">
  <workspace title="My blog" >
    <collection title="Entries"
      href="http://example.org/reilly/main" >
      <accept>entry</accept>
    </collection>
    <collection title="Pictures"
      href="http://example.org/reilly/pic" >
      <accept>image/*</accept>
    </collection>
  </workspace>
</service>
```

## Listing Collections

To retrieve the contents of a collection, send an authenticated HTTP GET request to the collection's URI.



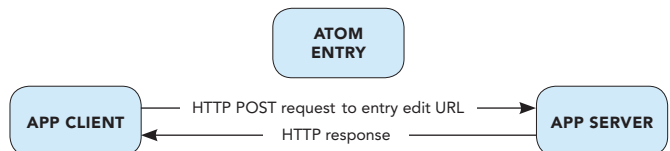
The server will respond by sending back an Atom feed containing the first portion of the collection and a next URI, which you can use to retrieve the next portion of the collection.

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <link rel="next"
    href="http://example.org/entries/60" />
  <link rel="previous"
    href="http://example.org/entries/20" />
  ...
  <entry> ... </entry>
  <entry> ... </entry>
  <entry> ... </entry>
  <entry> ... </entry>
  ...
</feed>
```

### Creating an Entry

To create a new entry within a collection, you simply post the XML for the entry to the collection's URI. For example, here's an example entry suitable for posting to an Atom server.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <id></id>
  <title>Atom test post title</title>
  <content>Atom test post content</content>
  <updated>2006-05-16T00:00:00Z</updated>
</entry>
```



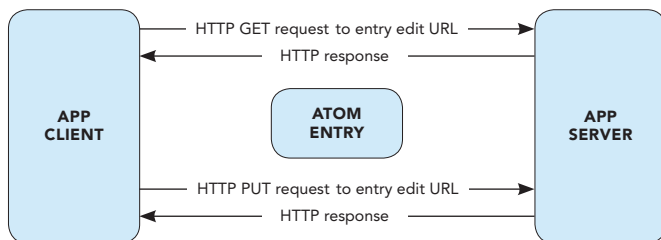
The server will respond by creating an entry based on what you posted. It will fill in some blanks, such as the ID, and will return the Atom entry as it appears on the server. It will add in an edit URI, as shown below in bold, which you can use to retrieve, update or delete the entry.

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <id>http://example.com/blog/entry/2223</id>
  <link rel="alternate" type="text/html"
    href="http://example.com/blog/entry/2223" />
  <link rel="edit" type="text/html"
    href="http://example.com/app/blog/entry/2223" />
  <title>Atom test post title</title>
  <content>Atom test post content</content>
  <updated>2006-05-16T00:00:00Z</updated>
</entry>
```

## The Atom Protocol, continued

### Updating an Entry

To update an entry, you first send an authenticated HTTP GET request to the entry's edit URI to get the latest copy of the entry. You then edit the entry and use an authenticated HTTP PUT to the edit URI to update it on the server.



## ATOMPUB IN THE WILD

Atom protocol has been spreading like wildfire since the specification was finalized in October 2007. Google and Microsoft have adopted it as the basis for many of their web services interfaces, for example:

**Google Data (GData)**—more than a dozen APIs that use Atom protocol plus extensions: <http://code.google.com/apis/gdata>

**OpenSocial**—access and manage Social Graph data via Atom protocol: <http://code.google.com/apis/opensocial>

**Microsoft Windows Live**—manage photos and store gadget data with Atom protocol: <http://dev.live.com>

**Microsoft ADO.Net Data Services**—use Atom protocol to access relational databases: <http://astoria.mslivelabs.com>

## ABOUT THE AUTHOR



### Dave Johnson

Dave Johnson is a North Carolina-based software developer who has worked in a variety of software companies including Rogue Wave, HAHT Software and SAS Institute. In 2002, unable to satisfy his urge to create cool software at work, Dave worked nights and weekends to create the open source, Java-based Roller blog server. Roller's popularity at [JRoller.com](http://JRoller.com) led to its adoption by Sun and IBM for internal and external blogs. It's now known as Apache Roller and is a project of the Apache Software Foundation. Dave now works as a Social Software Architect on the App Platform team at Sun Microsystems.

#### Publications

- *RSS and Atom in Action*, 2006
- *ProJSP*, 2002
- Article: *Building a J2EE Weblogger*, 2002

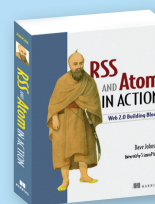
#### Projects

- Original developer of the Roller blog server
- Contributor to ROME feed parser/generator

#### Blogs

- *Blogging Roller*, personal weblog, 2002–present

## RECOMMENDED BOOK



*RSS and Atom in Action* offers clear, concise, and up-to-date coverage of all feed formats, web services protocols for blogging, and the very latest programming toolkits for Java and C# developers. It's the first book to cover the all-new IETF

Atom Publishing Protocol, the Java-based ROME feeds toolkit and Microsoft's new Feeds API for IE7 and Windows Vista.

## BUY NOW

[books.dzone.com/books/rss-and-atom](http://books.dzone.com/books/rss-and-atom)

Want More? Download Now. Subscribe at [refcardz.com](http://refcardz.com)

### Upcoming Refcardz:

- NetBeans IDE 6 Java Editor
- Ruby
- Groovy
- Core .NET
- Adobe Flex
- Apache Struts 2
- C#

### Available:

Published July 2008

- GlassFish Application Server
- Silverlight 2
- IntelliJ IDEA

### Available:

Published June 2008

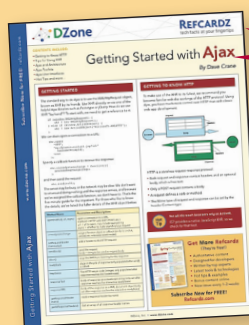
- jQuerySelectors
- Design Patterns
- Flexible Rails: Flex 3 on Rails 2

Published May 2008

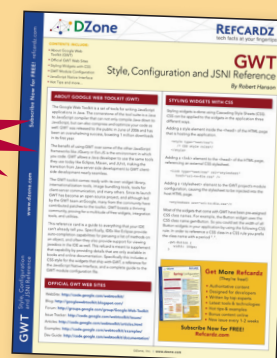
- Windows PowerShell
- Dependency Injection in EJB 3

Published April 2008

- Spring Configuration
- Getting Started with Eclipse



Getting Started with Ajax  
Published April 2008



GWT Style, Configuration and JSNI Reference  
Published April 2008



DZone communities deliver over 3.5 million pages per month to more than 1.5 million software developers, architects and designers. DZone offers something for every developer, including news, tutorials, blogs, cheatsheets, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZone, Inc.  
1251 NW Maynard  
Cary, NC 27513  
888.678.0399  
919.678.0300

Refcardz Feedback Welcome  
[refcardz@dzone.com](mailto:refcardz@dzone.com)

Sponsorship Opportunities  
[sales@dzone.com](mailto:sales@dzone.com)

ISBN-13: 978-1-934238-00-4  
ISBN-10: 1-934238-00-7



9 781934 238004



\$7.95