Introducing OpenSearch

July 24, 2007

Uche Ogbuji (/pub/au/84)

Search and web feeds go together pretty naturally, as anyone who has set up some kind of vanity search feed knows. You go to your favorite Web 2.0 search engine and set up a query like `http://web20.example.com/search=john+doe&ouptut=atom` and search for "john doe," but rather than getting back results as the usual HTML web page, you get it back in Atom format. You can subscribe to this URL in your favorite feed reader, and you have all the useful features of web feeds attached to this search query. Most notably, rather than having to poll the search engine yourself and having to remember which results you have seen, your reader will simply alert you when there are new results. This simple but very useful concept is the core idea behind the OpenSearch (http://www.opensearch.org) specification.

OpenSearch was originally developed at Amazon.com's A9 incubator. It's a specification under the Creative Commons Attribution-ShareAlike License, covering discovery and description documents for search engines, expression of queries, and the convention of RSS 2.0 or Atom Web feed results. It is very RESTful in nature and complementary to the Atom Publishing Protocol (APP). In fact, many have called for OpenSearch to serve as the query aspect of APP, which provides a way to access identified or located results, but no mechanism for ad hoc query. With all this affinity to Atom and REST, OpenSearch is a natural topic for this Agile Web column. OpenSearch 1.0 is still the latest full version; it has been around since 2005. Version 1.1 is in beta, but has some important improvements and is thus the version I'll be discussing.

Finding a Suitable Search Engine

Once you've found a search engine, the first issue is learning more about it and, in particular, how to query it. The OpenSearch description document format is designed to provide this information. Listing 1 is a simple example describing a fictitious search engine for XML.com.

Listing 1: OpenSearch description document

```
 <?xml version="1.0" encoding="UTF-8"?>

<OpenSearchDescription xmlns="http://a9.com/-/spec/opensearch/1.1/"

 xmlns:dc="http://purl.org/dc/elements/1.1/">

 <ShortName>XML.com search</ShortName>

 <dc:relation href="http://www.xml.com"/>

 <Description>Search XML.com articles and Weblogs</Description>

 <Tags>xml web</Tags>

 <Contact>admin@xml.com</Contact>

 <Url type="application/atom+xml"

  template="http://search.xml.com/?q={searchTerms}&p={startPage?}&format=ato

 <Query role="example" searchTerms="test+xml"/>

 <Attribution>All content Copyright 2007, O'Reilly and Associates</Attributi

</OpenSearchDescription>
```

It's pretty straightforward stuff for the most part. Elements such as `ShortName` and `Description` provide basic information for people who are browsing search engine information. `Tags` and `Attribution` offer additional details that are useful when narrowing down the choice to use the search engine. `Url` is an interesting element. It tells the search client how to query the search engine in terms of what URL forms can be used for searching. In this way it connects to another important section of the OpenSearch specification, URL template syntax, which I'll discuss in a later section. `Query` is another special element that, in this case, tells search clients that they can test the search engine (this test purpose indicated by `role="example"`) by querying with the search terms "test+xml." `Query` elements are more broadly used in OpenSearch results, as I'll discuss in a later section.

Foreign Markup

Listing 1 also demonstrates how you can extend OpenSearch description syntax using the common mechanism of adding foreign elements in a separate namespace. In this case, there is a Dublin Core metadata element `dc:relation` to express a simple relationship between *search.xml.com* and *www.xml.com*. It's interesting that, besides `Url` and `Query`, all the elements in Listing 1 could be expressed in equivalent Atom syntax. Even the foreign `dc:relation` is similar to `atom:link`, and the latter provides a bit more expressiveness (though you can even things up a bit by using Dublin Core qualifiers). Listing 2 is an example of the search engine description like in Listing 1, but converted to Atom syntax; it is purely the envelope with no entries, which is perfectly legal in Atom.

Listing 2: Atom document with the equivalent information to the OpenSearch description document in Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>

<feed xmlns="http://www.w3.org/2005/Atom" xmlns:os="http://a9.com/-/spec/open

  <id>http://search.xml.com</id>

  <link rel="self" href="http://search.xml.com"/>

  <link type="text/html" href="http://www.xml.com"/>

  <updated>2007-07-07T12:00:00Z</updated>

  <title>XML.com search</title>

  <subtitle>Search XML.com articles and Weblogs</subtitle>

  <author>

    <name>XML.com</name>

    <email>admin@xml.com</email>

  </author>

  <rights>All content Copyright 2007, O'Reilly and Associates</rights>

  <category term="xml"/>

  <category term="web"/>

  <os:Url type="application/atom+xml"

   template="http://search.xml.com/?q={searchTerms}&p={startPage?}&format=ato

  <os:Query role="example" searchTerms="test+xml"/>

</feed>
```

There is no need for Dublin Core, in this case, given `atom:link`. But rather than abuse that element, `Url` is pulled in from the OpenSearch namespace to express the search URL template. The purpose of this example is not to disparage OpenSearch's choice in rolling its

own format. I do believe that it's useful to reuse formats where possible, but I also think that it's important not to push reuse until you're stretching a format to an alien purpose. One could make an argument that Listing 2 stretches the purpose of Atom syntax too far.

URL Templates

Take another look at the `Url` element in Listing 1, which serves as the mechanism for telling the search client how to query the search engine. The `template` attribute looks like a URL, but the parts within curly braces are parameters the client provides to specialize the search. There are half a dozen parameter names like `searchTerms` with a purpose established within the OpenSearch spec. The `searchTerms` parameter is a placeholder for the search criteria (e.g., john+doe); `startPage` allows the client to specify a page within the result set. More on result pages in a later section. You can use a parameter in the form of an XML QName for a foreign namespace to cover meanings not provided by the standard parameters. Notice the question mark after the `startPage` parameter in Listing 1. This means the search client is free to not provide a value for this parameter (i.e., to substitute it with an empty string). The search client must provide a non-empty value for `searchTerms` because it does not have the question mark.

Result Format

Once again, a central idea of OpenSearch is that search results come as web feeds. The supported formats are RSS 2.0 and Atom 1.0. In this article, and in my personal recommendation, I stick to the latter. Each search result corresponds to an Atom entry, using the usual semantics for entry syntax. There is, however, some interesting specialization at the feed element level. Listing 3 is a sample OpenSearch query result with a single result item.

Listing 3: OpenSearch Atom search result

```
<?xml version="1.0" encoding="UTF-8"?>

<feed xmlns="http://www.w3.org/2005/Atom" xmlns:os="http://a9.com/-/spec/open

  <id>urn:uuid:60a76c80-d399-11d9-b93C-0003939e0af6</id>

  <title>XML.com search: atom store python</title>

  <link rel="self" type="application/atom+xml"

        href="http://search.xml.com/?q=atom+store+python&p=&format=atom"/>
```

```
  <link rel="alternate" type="text/html"

        href="http://search.xml.com/?q=atom+store+python&p="/>

  <link rel="search" type="application/opensearchdescription+xml"

        href="http://example.com/opensearchdescription.xml"/>

  <os:totalResults>1</os:totalResults>

  <os:startIndex>1</os:startIndex>

  <os:itemsPerPage>10</os:itemsPerPage>

  <os:Query role="request" searchTerms="atom store python" startPage=""/>

  <updated>2007-07-07T12:00:00Z</updated>

  <author>

    <name>XML.com</name>

    <email>admin@xml.com</email>

  </author>

  <rights>All content Copyright 2007, O'Reilly and Associates</rights>

  <entry>

    <title>XML.com: Implementing the Atom Publishing Protocol</title>

    <!-- Note: following URL modified for article formatting reasons -->

    <link href="http://www.xml.com/pub/a/2006/07/19/implementing-app-python-

    <id>http://www.xml.com/2006/07/19/implementing-app-python-wsgi</id>

    <updated>2006-07-19T15:00:00Z</updated>
```

```
    <content type="text">

     Joe Gregorio's latest Restful Web column implements the Atom Publishing

     Python web service using WSGI.

    </content>

    <author>

      <name>Joe Gregorio</name>

    </author>

  </entry>

</feed>
```

The links with `rel="self"` and `rel="alternate"` follow the usual Atom semantics. The `rel="search"` link is a convention added by OpenSearch for feed auto-discovery. When accessing this URL you should get a search engine description document like Listing 1. Notice the `application/opensearchdescription+xml` media type, which OpenSearch proposes for description documents. You can also use special link types for paging search results. If a search would produce thousands of results, neither the client nor the service provider is likely to want to pile them all into a single result feed document, especially considering that most search engines provide hits of most likely interest in early pages. The Feed Paging and Archiving (aka Feed History) extension to Atom provides a simple mechanism for breaking down large virtual feeds into pages or sections in such cases. It's currently an IETF Internet Draft, but probably will be adopted as a standard soon. OpenSearch adopts its conventions for paging search results. An OpenSearch response might be one of a series of feeds, each of which represents a subset of the total results, including links with types such as `first`, `last`, `prev`, and `next` to inform the search client how to navigate through the results. Listing 3 also shows elements, `totalResults`, `startIndex`, and `itemsPerPage` in the OpenSearch namespace that provide additional contextual metadata for search results. The common URL parameter `startPage` allows the search client to jump to a specific result page, and the `count` parameter controls the number of result items per page.

OpenSearch provides a few conventions to work with HTML web pages, including meta tags for auto-discovery of description documents and the `totalResults`, `startIndex`, and `itemsPerPage` information about search results.

Searching, Agile Web Style

OpenSearch really just provides the framework of a query mechanism to complement the Atom Protocol. It defines enough semantics to tell you how to express simple full-text searches. You can extend it for more specialized query by adding your own extension parameters in URL templates. For example, you might want to specify a parameter to limit searching to a specific element in Atom feeds using a template like `http://search.xml.com/?q={searchTerms}&f={x:restrictField?}` (you'd have to define a namespace for the "x" prefix). This would allow the search client to search for "xslt" within summaries by specifying `http://search.xml.com/?q=xslt&f=atom:summary`. By keeping it simple, OpenSearch complements other related technologies very well, and adheres to solid Agile Web principles. There is a long and growing list of OpenSearch tools and search engines, so there is a good chance this specification will guide how we approach search and query for Web 2.0.
