At the Forge

Syndication with RSS

*Syndication isn't only for blogs.*

by Reuven M. Lerner

When I first started to use the Web, anyone who put up a site would send e-mail to Tim Berners-Lee, giving
the URL and a brief description of what the site was about. Tim would respond with a brief personal note and
would update his master list of Web sites, which anyone with a browser could retrieve. Active participants in
the Web community would review that list regularly—and its successor, published by the same people who
produced the Mosaic browser—for new and updated sites, so as not to miss a bit.

Fast-forward more than a decade, and the Web is obviously too large for anyone to maintain a list of new
sites manually. And even if that were possible, no one can read more than a fraction of the new content that
goes live each day. Add to this the fact that now there are hundreds of thousands of Weblogs, or blogs, many
of which are frequently updated, and the task becomes even more difficult.

One solution is to use your browser's bookmarks. But after a while, it becomes a chore to check bookmarks
each day, let alone several times a day. It would be nice if each site could indicate when its content has
changed, so that you would visit only when necessary.

This insight is not new; the idea of announcing changes to Web content has existed for several years. But I
must admit, it was only a few months ago when I began to realize how behind the times I was, when I would
start each day by visiting a few of the sites in my bookmarks. By taking advantage of an RSS
aggregator—namely, a program that looks at the RSS feeds from various sites and alerts me when there has
been an update—I am able to do more in less time.

This month, we discuss the popular RSS (really simple syndication or RDF site summary) family of formats,
looking at ways in which it might be useful and how it is created.

Simple RSS

RSS began as the brainchild of Netscape, the Internet software company that has since been absorbed (and
largely dismembered) by AOL. Netscape wanted to offer people news from multiple sources but on a single
page. They accomplished this by publishing the specification for RSS 0.90. Anyone interested in publishing
news through the Netscape portal needed to do so in RSS. Netscape's system would retrieve this RSS
document from the Web site in question and publish the results.

Although RSS 0.90 sparked a revolution, it also was fairly complicated. Dave Winer, then the head of
Userland Software, turned RSS into a simple specification, renamed it RSS 0.91 and began to talk about it on
his Weblog, scripting.com. Suddenly, RSS 0.91 was everywhere; Dave's orange XML buttons, indicating that
you could get an RSS feed from a site, became quite popular. Within a few years, RSS feeds sporting other
versions were available as well. RSS 1.0 was developed by a group of developers on the Web, and RSS 2.0,
coordinated by Dave, was seen as an upgrade to 0.9x.

If you have been following this history, you might have reached the conclusion that now there are three
different syndication formats called RSS. Aside from the version numbers, and some obvious similarity between the different versions, these are three different formats.

In many ways, RSS resembles HTML and HTTP, which began as simple-to-understand, simple-to-implement standards written by a small group of people. All three of these standards have been forced to mature quite a bit in the past few years, losing some of their flexibility and simplicity in the process.

RSS 0.91 is the simplest of the bunch and is still rather popular. Everything sits within an <rss> element, which identifies its version and contains a single <channel> element. Several required tags (title, link, description, language and image) are followed by one or more <item> elements. Each item has its own title, link and description. For example, here is a simple RSS feed from my Weblog:

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN" "http://my.netscape.com/publish/formats/rss-0.91.dtd">
/rss version="0.91">
 <channel>
   <title>Altneuland</title>
   <link>http://altneuland.lerner.co.il/</link>
   <description>Reuven's Weblog</description>
   <item>
     <title>Independence Day</title>
     <link>http://altneuland.lerner.co.il//40</link>
   </item>
   <item>
     <title>Linux desktops for the masses? Ha!</title>
     <link>http://altneuland.lerner.co.il//39</link>
   </item>
 </channel>
</rss>
```

If you examine the above RSS feed, you can see it does not conform to the RSS 0.91 specification I described previously. Specifically, it lacks the required language and image elements within channel, and it lacks a description element within each item. Unfortunately, this comes as no surprise; as was the case with HTML in its earliest years, software authors often cut corners, producing output that was good enough for most purposes. And indeed, COREBlog (which, as of this writing, I am using to produce my Weblog) seems to have cut such a corner, producing a usable, but substandard, RSS 0.91 feed.

### Producing a Feed

If you want to produce a legitimate RSS feed, you probably should use one of the many open-source modules available for most popular languages. For example, Perl developers can use the XML::RSS module, available from any CPAN mirror (see the on-line Resources section).

To create an RSS feed with this module, we can write a simple program that looks like this:

```perl
#!/usr/bin/perl
use strict;
use diagnostics;
use warnings;
use XML::RSS;
```
my $url = "http://altneuland.lerner.co.il/";

my $rss = new XML::RSS (version => '0.91');
$rss->channel(title => 'Altneuland',
   link => $url,
   language => 'en',
   description => "Reuven Lerner's Weblog");

$rss->add_item(title => 'Being scared',
   link => "$url/43/index_html",
   description => 'Blog entry');

print $rss->as_string;

We begin the program with the creation of a new XML::RSS object, specifying that we want to use version 0.91 of the RSS standard. We then specify the individual items we want to define, and we can omit the image tag. Although the XML::RSS module allows us to omit any or all of the tags from our channel descriptor, it would not make sense to leave out some of them, such as title and link.

We then add individual items to the channel, one by one, until we have completed all of them. At that point, we can produce the RSS output, which appears as follows:

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN" "http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
  <channel>
    <title>Altneuland</title>
    <link>http://altneuland.lerner.co.il/</link>
    <description>Reuven Lerner's Weblog</description>
    <language>en</language>
    <item>
      <title>Being scared</title>
      <link>http://altneuland.lerner.co.il/43/index_html</link>
      <description>Blog entry</description>
    </item>
  </channel>
</rss>

Most programs that produce RSS feeds are not going to invoke $rss->add_item(), as I did above, on a case-by-case basis. If we are syndicating a Weblog, commercial news feed or other frequently updated site, we probably would create an RSS feed by looping over a set of files in a directory or (better yet) over rows in a relational database.

For example, the following code fragment would retrieve all of the Weblog entries posted within the last 24 hours to a hypothetical weblog_entries table in PostgreSQL:

# Get all entries from the latest 24 hours
my $sql = "SELECT entry_id, title, link, description
           FROM weblog_entries
           WHERE when_entered >= (NOW() - interval '1 day')";

# Prepare the SQL statement
my $sth = $dbh->prepare($sql);

# Execute the SQL statement
my $result = $sth->execute;

# Iterate through resulting rows
while (my $rowref = $sth->fetchrow_arrayref)
{
  my ($id, $title, $link, $description) = @$rowref;

  $rss->add_item(title => $title,
                 link => $link,
                 description => $description);
}

This demonstrates one of the many advantages of storing a Weblog in a relational database. Once the entries are stored in a database, it is easy to add new functionality, such as syndication. Although XML::RSS provides functionality (and sample code, in its perldoc on-line documentation) for limiting the number of syndicated articles to a set number, this seems like a much more appropriate job for a database, where the LIMIT modifier can set a maximum number of returned rows.

## Moving to RSS 1.0

RSS 1.0 was a reaction to RSS 0.91, tying it more closely to various World Wide Web Consortium (W3C) standards, including RDF. The version numbers might have you believe that 1.0 was an upgrade to 0.91; however, the two are (unfortunately) independent and uncoordinated. 0.91 (and its successor, RSS 2.0) have been authored by Dave Winer based on input from the developer community, and 1.0 was written by an open consortium of developers. RSS 0.91 and 2.0 have more in common than 1.0 does with either of them, which, not surprisingly, has led to some confusion.

RDF, the resource development framework defined by the W3C, is part of the semantic Web project, which wants to make the Web understandable to computers as well as by people. This requires standardizing the metadata, or invisible descriptions that accompany the output from a site. RDF is one attempt at such a standardization.

RSS 1.0 thus tied syndication to RDF, adding the use of XML namespaces along the way. XML namespaces allow us to combine different XML definitions into a single document.

To create a syndication feed that complies with RSS 1.0, we need to make only a simple change to our program from above, changing the version number in our invocation of new on XML::RSS:

```perl
my $rss = new XML::RSS (version => '1.00');
```

And indeed, if we make this change, the resulting RSS feed looks slightly different:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns="http://purl.org/rss/1.0/"
  >
<channel rdf:about="http://altneuland.lerner.co.il/"/>
```

<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns="http://purl.org/rss/1.0/"
  >
<channel rdf:about="http://altneuland.lerner.co.il/"/>
There are several things to notice in this output, beginning with the definition and use of several namespaces, introduced with the xmlns attributes, and then with the use of additional RDF-specific attributes, such as rdf:about and rdf:resource.

But the above doesn't do justice to RSS 1.0, which allows us to specify a great number of other parameters. For example, we can set information about our site's frequency of syndication updates by adding a syn section to our invocation of $rss->channel(); RSS 1.0 also includes support for Dublin Core, an increasingly popular and standard method for tagging documents.

RSS 2.0

The good news, as we have seen, is that RSS 1.0 is not significantly more difficult to create or parse than RSS 0.91 was, assuming you are using decent tools. However, the complexity of RSS 1.0 was seen as unnecessary by some.

Indeed, after several attempts to reach a consensus on RSS 1.0, a number of developers banded together to work on something now known as Atom. Although discussion of Atom will have to wait until next time, this spurred the RSS camp (led by Winer) to produce RSS 2.0.

You can produce RSS 2.0-compatible feeds by changing the version number in the invocation of new:

my $rss = new XML::RSS (version => '2.0');

You must say 2.0; neither 2 nor 2.00 will work, because the version check uses a string comparison, rather than a numeric one.

What does RSS 2.0 look like? Well, you might be surprised:

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

<rss version="2.0"
     xmlns:blogChannel= "http://backend.userland.com/blogChannelModule">

  <channel>
    <title>Altneuland</title>
    <link>http://altneuland.lerner.co.il/</link>

    <item rdf:about= "http://altneuland.lerner.co.il/43/index_html">
      <title>Being scared</title>
      <link>http://altneuland.lerner.co.il/43/index_html</link>
      <description>Blog entry</description>
    </item>

  </channel>

</rss>
This looks a lot like RSS 0.91 and, thus, seems like a stripped-down version of RSS 1.0. But when we remember that RSS 2.0 is a successor to 0.91 and is designed to fix some of its flaws while remaining small, simple to implement and flexible, it becomes more obvious.

RSS 2.0 includes a number of improvements over 0.91, most especially the idea of using namespaces as modules that add new functionality. RSS 2.0 doesn't define or use nearly as many namespaces as 1.0 does, but that's because it is not trying to implement RDF.

Partly because of criticism that he personally held the copyright to the RSS 2.0 specifications, Winer gave ownership to Harvard University. It is assumed that Winer will continue to play a major role in the development of RSS 2.0, but he will no longer be the final arbiter regarding usage or extensions.

However, the split seems to be final; there is now an Atom camp and an RSS camp, and I find it hard to believe they will meet. But given the conflicting goals they have set for themselves, this should not come as a surprise—after all, you cannot expect to have flexibility and ease of implementation in the same specification.

**Conclusion**

This month, we looked at the different flavors of RSS currently in use and compared their different styles and project goals. Luckily, someone who wants to produce a bare-bones syndication feed does not need to work very hard. Although programmers can add some version-specific fields, the basics are the same for all versions of RSS, even those that are ostensibly incompatible. The resulting RSS feeds, of course, can look quite different, depending on which version is used.

Next column, we will look at the upstart Atom syndication format, which is growing rapidly as a competitor to RSS. Once we have done that, we will look at how to build our own news aggregator, allowing us to interpret and work with syndication feeds from a wide variety of sources. We also will consider different ways in which RSS can be used and how aggregators can provide more than the latest news and opinions.

**Resources for this article:** [www.linuxjournal.com/article/7702](http://www.linuxjournal.com/article/7702).

Reuven M. Lerner, a longtime Web/database consultant and developer, now is a first-year graduate student in the Learning Sciences program at Northwestern University. His Weblog is at [altneuland.lerner.co.il](http://altneuland.lerner.co.il), and you can reach him at [reuven@lerner.co.il](mailto:reuven@lerner.co.il).