Push RSS to new limits

Use RSS to construct an associative database

Skill Level: Intermediate

Jonathan Levin
Trainer and consultant
Freelance

18 Dec 2007

This tutorial presents an innovative use of the well-known Really Simple Syndication (RSS) format’s associative properties to emulate the functionality of a simple relational database. It demonstrates using RSS channels to store contact information and meeting information—much as a personal address book and calendar does. It uses RSS elements and attributes such as items and guids to create a neural-network-like mesh of related data.

Section 1. Before you start

This tutorial is intended for people who want to better understand the well-known Really Simple Syndication (RSS) format and what it can do. Specifically, it shows how to leverage some of the lesser-known features of RSS to provide better data storage.

This tutorial assumes that you understand the basic concepts of XML, but familiarity with RSS is not required. Programming examples use PHP, but the basic concepts apply for any programming language.

About this tutorial

Many people know that RSS is the format used for most syndication purposes, such as site feeds, but in the past few years it’s also become the storage format of choice
for other applications, such as data distribution, contact management, and other purposes. By storing data as RSS, you make it possible to use RSS aggregators and readers to access that data.

All of this leads to the notion of using RSS to store information that previously might be stored in a relational database.

The tutorial begins with an explanation of the RSS format and some actual feed samples. It then takes a twist on the classic format, and redefines it for your alternative purpose: storing arbitrary data items and their associations. You define your database format with three tables, and proceed to query them, showing that you can both emulate traditional SQL SELECT, and also provide an associative lookup. Lastly, you learn how to handle the query results, leaving them in XML form, or transforming them to user-presentable content using XML StyleSheet Transformation Language (XSLT).

Prerequisites

The following tools are needed to follow along with this tutorial:

- **PHP**: Any version of PHP will do nicely. This tutorial uses PHP's back-end textual processing rather than the traditional Document Object Model (DOM) to process the XML.
- **Your favorite XML editor** is required to edit XML and XSLT files. The author uses vim on UNIX®, and Notepad on Windows®.

Section 2. RSS overview

This section begins with an explanation of the RSS format and provides a sample feed. It also describes all the elements of an RSS feed as well as explains the use of RSS as a data repository.

What is RSS?

RSS is a format that is widely adopted to syndicate or publish Web site and other content. Unlike the HTML view of the site, RSS summarizes a Web site as an XML channel with any number of items. In other words, RSS is really just a particular grammar of XML. Using a few basic elements and a rather simple schema, it allows for the encapsulation of item elements within a channel element. While these items
are normally information such as blog entries, in this context, the items can be anything you want them to be: images, URLs, clips, or textual data. The channel is the feed source, which serves as a collection of the items it contains, as well as provides metadata common to all items. In object-oriented terms, you might say the channel object contains an array of zero or more item objects.

An increasing number of sites now offer RSS feeds: From online newspaper sites (like the New York Times), through online Bulletin Boards (Craig's List), comic strips (Dilbert, XKCD) and more. RSS support has also become ubiquitous in all browsers (being widely adopted in FireFox, Opera and Safari, and in Internet Explorer 7). MacOS X and Vista both attempt to better integrate RSS with the user's desktop through widgets and gadgets.

Summarizing arbitrary content in XML form opens up many possibilities. The most significant of them is that now specialized applications (called RSS aggregators) can automatically process the content with greater ease. Rather than parsing HTML, which is an (as yet) impossible task to perform efficiently, the RSS format provides a machine-friendly XML. XML is far easier to process automatically, thanks to its meta-data tags. This allows the aggregator to focus on the individual content items, as well as somewhat deduce their relevance to the reader's interest by keyword classification.

Sample RSS feed

What makes RSS even more powerful, however, is not the simple yet elegant schema, but the fact that it can easily be extended by putting just about any form of item in the channel. This is done by including any additional XML namespace in the RSS element. This is best demonstrated by an example, Yahoo's Weather feed, shown in Listing 1, with annotations.

**Listing 1. Sample RSS feed**

Yahoo's Weather report for zip code 02139

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<rss version="2.0"
     xmlns:yweather="http://xml.weather.yahoo.com/ns/rss/1.0"
     xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#">
    <channel>
        <title>Yahoo! Weather - Cambridge, MA</title>
        <link>http://us.rd.yahoo.com/dailynews/rss/weather/Cambridge__MA/</link>
        <description>Yahoo! Weather for Cambridge, MA</description>
        <language>en-us</language>
        <lastBuildDate>Wed, 28 Nov 2007 2:54 pm EST</lastBuildDate>
        <ttl>60</ttl>
        <yweather:location city="Cambridge" region="MA" country="US" />
        <yweather:units temperature="F" distance="mi" pressure="in" speed="mph" />
        <yweather:wind chill="38" direction="350" speed="9" />
        <yweather:atmosphere humidity="43" visibility="1609"
                        pressure="30.38" rising="1" />
        <yweather:astronomy sunrise="6:50 am" sunset="4:14 pm" />
    </channel>
</rss>
```
In this way, an RSS-aware application that also has built-in weather detection elements can opt to either display the description (which, as you can see, is an HTML snippet that can be embedded in code) or isolate only the items of interest (for example, yweather:forecast for day="Thu"). As more grammars are published and more semantic information is added, you can expect RSS to increase in use, perhaps one day becoming a content distribution platform as dominant as HTML is today.

It's this flexibility that enables you to consider using RSS as a generic data storage format. But first, let's start with a refresher on the information that RSS provides.

Elements in an RSS feed

Table 1 summarizes the important elements you'll find in an RSS feed. Bolded elements, where shown, are optional. Otherwise, elements are required. Most elements can be used either in the <channel> context, or in the <item> context,
allowing the feed publisher to set defaults for the channel, yet override them on a
per-item basis, if required.

Table 1. Important elements in an RSS feed

<table>
<thead>
<tr>
<th>Element</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A sentence or two describing the channel and used in the tooltip of an RSS reader when the cursor hovers over the channel name.</td>
</tr>
<tr>
<td>Link</td>
<td>URL of the channel (HTML content).</td>
</tr>
<tr>
<td>Title</td>
<td>Display title of this RSS channel and used as the channel name.</td>
</tr>
<tr>
<td>Image</td>
<td>Optional image child element. Image has Height, Width, Title, Link, URL children, which are rendered as:</td>
</tr>
<tr>
<td>pubDate</td>
<td>RFC822 time that specifies when the content was published</td>
</tr>
<tr>
<td>ttl</td>
<td>Suggested time, in seconds, for caching this channel's content.</td>
</tr>
<tr>
<td>category</td>
<td>The taxonomy of this particular channel, or item.</td>
</tr>
<tr>
<td>guid</td>
<td>Globally Unique Identifier. This is a URL or other identifier that is guaranteed to be unique, so as to provide an accurate and single reference to the item in question.</td>
</tr>
</tbody>
</table>

The optional elements, in particular, add powerful functionality to RSS. pubDate and ttl (Time To Live) serve to indicate when and whether the content is live, enabling a reader to cache the content on the one hand, but avoid stale content on the other. The guid is any globally unique identifier, and you'll make use of that in our application. Most applications use a URL here. It doesn't necessarily have to be a tangible, real URL so long as it is not quoted anywhere else in any context but that of the item in question. The category element is key in deducing the semantic context of the content. If you've seen sites such as flickr, Digg and others that use tags, you've seen categories in action.

Table 1 is but a partial, yet concise listing. You can find a full listing of the elements
in the RSS specification (see Resources for a link). Recall, also, that you can add just about any element, so long as you declare its namespace in the RSS container context.

Using RSS as a data repository

So far there is nothing new. RSS has been presented like any spec. And chances are most of you already use RSS feeds in their personal aggregators, either in specialized readers or using built-in browser support. But here comes the twist: You can use RSS as an actual data storage format.

In the traditional, classic model of a database, data is stored in tables. Tables are indexed by one or more keys, and items are stored as records in the tables. Using any structured query language (most often, SQL), you can retrieve any record or field thereof from a specific table, and link (or JOIN) it to data in other tables. The model shown in this tutorial is a far more associative one, mimicking the thought processes of the human brain. Before going on, let's understand how association works.

The brain is a complex mesh of cells (or neurons) connected to one another by tendrils (axons). While an accurate description of its workings is well beyond the scope of this tutorial (and actually does not exist, at the time of writing), consider a simplified model wherein each neuron is really an RSS item. This item has a title, possible description, and, most importantly, a unique representation in its guid. The item may optionally have all sorts of specific data associated with it; let's leave that opaque at the moment. Now, this item can be linked with other items by means of axons, which are in your case links. What do you get? A meshed topology of items and links, which is also called, an associative network.

Why bring this up? Because in this tutorial, you will build an application in which disparate pieces of data—people, places, and appointments—are linked to each other through RSS data much the way neurons are linked to each other through axons.

Next let's build the database.

---

Section 3. Building an RSS database by example

The concept of using RSS as a data repository, as defined in the previous section, is best illustrated by an example, which you will create in this section of the tutorial. You will create three test feeds:
• The directory, which includes information on individual people.
• A map of places.
• A calendar of events. The events define things the people will do in specific places, and serves as a way to link together the two sets of data.

People to meet

To define contacts, you can use any XML format you want. One option is the XMLized form of VCF, which was demonstrated in an earlier tutorial entitled, "Expand the editing capabilities of OpenOffice with XSLT" (see Resources for a link). You'll define your address book as follows:

Listing 2. Sample RSS database: Your directory

```xml
<rss version="2.0"
 xmlns:mesh="http://www.hisown.com/NameSpaces/Mesh/20071203">
  <channel>
    <title>People</title>
    <item>
      <guid>Simpsons:/Homer_J_Simpson</guid>
      <title>Homer Jay Simpson</title>
      <link>http://en.wikipedia.org/wiki/Homer_Simpson</link>
      <description>Nuclear Plant Drone, Sector 7-G</description>
      <xvcf:.../>
      <mesh:axon type="relation" value="wife">Simpsons:/People/Marjorie_M_Simpson</mesh:axon>
      <mesh:axon type="relation" value="son">Simpsons:/People/Bartholomew_J_Simpson</mesh:axon>
      <mesh:axon type="relation" value="father">Simpsons:/People/Abraham_A_Simpson</mesh:axon>
      <mesh:axon type="location" value="work">Simpsons:/Places/Springfield_Nuclear_Power_Plant</mesh:axon>
    </item>
  </channel>
</rss>
```

The directory

The test feed is attached to this tutorial. You can also view it online: http://hisown.com/rss/simpsons_contacts.xml.

This test feed is a fairly simple one. You compose an RSS channel as a container for your items. Then you add the items, each item with a guid, which is really the most important element, as it is used for linking. Then you add a title, which is used for display purposes, and the link, which is a link to the Wikipedia entry. (Imagine this is a link to the person's bio and details). The <xvcf:> namespace element can be anything, and is in fact shown there as a placeholder to indicate that additional item information can be stored here. Then, you use a new namespace, appropriately called mesh. Your links, or connectors, are the axon elements, and you can tag each
with a type and value, although you won’t make use of them in this tutorial.

Actual semantic networks make use not only of the links between elements, but their direction (a'b or b'a), type, and value as well.

Places to go

To define locations, you create a second channel, as shown in Listing 3.

Listing 3. Sample RSS database the map

```xml
<rss version="2.0">
  <channel>
    <title>Places</title>
    <item>
      <guid>Simpsons:/Places/Springfield_Nuclear_Power_Plant</guid>
      <title>Springfield Nuclear Power Plant</title>
      <description></description>
      <image>
        <url>http://upload.wikimedia.org/wikipedia/en/6/6e/Snpp.gif</url>
        <title>Springfield Nuclear Power Plant</title>
      </image>
      <!-- additional details about this location, if any !-->
      <mesh:axon type="relation" value="proprietor">
        Simpsons:/People/Charles_Montgomery_Burns
      </mesh:axon>
    </item>
  </channel>
</rss>
```

If you see this channel as similar to the other, it makes sense. Your RSS channels essentially all share the same format. The actual data in the items is next to irrelevant! What matters is the item guid.

Errands to run

To define events for your fictional characters to experience in their fictional places, you will create a third channel. These events serve to link the people and places, as shown in Listing 4.

Listing 4. Sample RSS database the calendar

```xml
<rss version="2.0">
  <channel>
    <title>Calendar</title>
    <item>
      <guid>http://jade.hisown.com/Calendar/Meetings/1234</guid>
      <title>Meeting with Mr. Burns</title>
      <description></description>
      <link>
        <!-- additional details about this event !-->
      </link>
    </item>
  </channel>
</rss>
```
Again, nothing new here. You have your engagement, in this case a meeting for Homer with the Boss, in a specified location. Notice all the axons are connected to already predefined elements in either the /People/ or the /Places/ namespaces.

You now have an associative link by means of the <mesh:axon> element. Note how you have defined for each such link a link type (relation or location) and value (father, son, participant). This set of definitions or classifications (people, places) is called a taxonomy. The building of relationships (father, son, is-a, subclass-of) is often referred to as an ontology.

Notice also that you link the data together using RSS's built-in data. The value of the axon element refers back to the guid, which uniquely defines the RSS item to which it refers.

The very basic application of a taxonomy is demonstrated here. You can extend this much further: Use specific XML grammars, relationships and associations between items of a similar nature, leading to the formation of what's known as the Semantic Web (or Web 3.0, even though it isn’t quite clear what Web 2.0 is). Several initiatives already in progress for that, most notably the W3C’s Web Ontology Language (see Resources for a link).

Your sequitur will be to provide some type of interface to your data store. But for that to happen, you first need to find a way to easily manipulate the RSS data programmatically.

Now that you have the data, you need a way to manipulate it.

Section 4. Parsing RSS content

RSS content is essentially just another XML dialect. There are many ways to parse
XML, with many complex libraries to handle DOM and/or SAX processing, for every language. PHP is not short in any of these respects. But you'll go with a simplified approach of parsing the RSS using PHP's powerful support of regular expressions. Yes, regular expressions look like curses, but their power over text is second to none.

**Using regular expressions**

In this tutorial, you'll use regular expressions as a quick and easy alternative to the various methods of parsing, as they are built-in and have a simple usage mechanism—which cannot be said for PHP's DOM libraries. Now, DOM buffs are ready to protest, boasting of DOMs obvious advantages—full Document tree, schema validation capabilities, insensitivity to whitespace, and much more. But, since this tutorial already demonstrates one avant garde concept, why not pack two for the price of one? DOM and SAX both have to somehow parse the underlying textual XML. And that somehow is through, you've guessed it -- regular expressions.

Those of you who have tried regular expressions in the past, must be going "@#$%^" (then take even that back, realizing it also is a regular expression :). But, you'll keep your usage of regular expressions to a bare minimum and wrap it in a single function, which will fetch elements for you.

**Creating your element fetcher**

Using a function to wrap the RegExp logic makes sense, as it abstracts the actual implementation of getting the element (using the RegExp) from the interface ("get me that element"). As an exercise, the faithful reader might consider rewriting this function by using the DOM.

Enter the following function, to get specific XML elements from any document, or feed, into an accessible PHP file (see Listing 5).

**Listing 5. PHP function to retrieve element from an XML document**

```php
function get_elements($xml, $element, $multiple)
{
    if ($multiple) {
        /* Return ALL matches of this element in the $xml provided.
         * Results will be returned as an array or arrays
         */
        if ($multiple) {
            $item = preg_match_all("<$element(|[^>]*)>(.*)</$element>#msU",
                $xml,
                $matches,PREG_PATTERN_ORDER);
            /* Trim whitespace. This could've been done in the RegExp, */
            /* but you can also use PHP's trim() function for that. */
            for ($i = 0;
                $i < count ($matches[2]);
```
$i++;
    } else {
        $item = preg_match("/<$element([^>]*)>(.*)</$element>/msU", $xml, $matches);
        $matches[2] = trim($matches[2]);
    }

    /* When you get here, $matches is either an array of arrays, or a 
     * single array, in the following format: 
     * [0] => Entire pattern matched 
     * [1] => "", or attributes of this element (if any) ([^>]*) 
     * [2] => Element content (corresponding to (.*)) 
     * You thus choose to return [2] = Element contents, or array of 
     * matched elements' contents. */
    return ($matches[2]); /* Element content */
}

As you can probably tell, the trick here is in the crafting of just-the-right-regular-expression. In your case it is: 
#<$element([^>]*)>(.*)</$element>#msU.

The actual RegExp is bounded by an arbitrary character [usually slash (/) by convention, but since you need slash (/) in the expression itself, you're better off to select the pound sign (#)]. You can break it down as follows in Table 2.

### Table 2. Element RegExp disassembled

<table>
<thead>
<tr>
<th>RegExp portion</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;$element</td>
<td>The Element opening tag—element name. Here less than (&lt;) is a literal less than (&lt;), and $element will be interpolated by PHP.</td>
</tr>
<tr>
<td>([^&gt;]*)</td>
<td>Either nothing, or anything but a closing greater than (&gt;).</td>
</tr>
<tr>
<td>&gt;</td>
<td>The closing greater than (&gt;) of the element</td>
</tr>
<tr>
<td>(.*</td>
<td>The element content</td>
</tr>
<tr>
<td>&lt;/$element&gt;</td>
<td>The element closing tag</td>
</tr>
</tbody>
</table>

Following the expression are the following behavioral modifiers shown in Table 3.

### Table 3. Element RegExp Modifiers explained

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Empty state. Increment after the preceding character.</td>
</tr>
<tr>
<td>b</td>
<td>Matches an empty character.</td>
</tr>
<tr>
<td>s</td>
<td>Matches a space character.</td>
</tr>
<tr>
<td>m</td>
<td>Multi-line. Matches an end of line.</td>
</tr>
<tr>
<td>u</td>
<td>Unicode mode.</td>
</tr>
<tr>
<td>s</td>
<td>Matches a space character.</td>
</tr>
<tr>
<td>m</td>
<td>Multi-line. Matches an end of line.</td>
</tr>
<tr>
<td>u</td>
<td>Unicode mode.</td>
</tr>
</tbody>
</table>

Push RSS to new limits © Copyright IBM Corporation 1994, 2008. All rights reserved. Page 11 of 22
Now let's incorporate this functionality into the overall application.

**Putting it all together**

So now, parsing an RSS feed to get at the data becomes straightforward. Wrap the call to `get_elements()` as in Listing 6.

**Listing 6. PHP function to retrieve a specific element by child node and value**

```php
function get_specific_element($feed, $child, $value) {
    $items = get_elements($feed, "item",true);
    if (!$items) { echo "What are you feeding me? No items here!\n"; return(false);}
    for ($i = 0 ; $i < count ($items);$i++){ /* to get child elements, simply call on parent element, * with child name as argument */
        $val = (get_elements($items[$i],"$child",false));
        if ($val == trim($value)) {return ($items[$i]); } ;
    }
    return(false);
}
```

And to call this function, all you need to do is use Listing 7.

**Listing 7. Processing your RSS database using PHP**

```php
$feed = get_channel ($feed_url);
$desired_child = /* Whatever, obtain from user */;
$desired_value = /* Whatever, obtain from user */;
$elem = get_specific_element($feed, $desired_child, $desired_value);
/* Have some fun with this element: */
$title = (get_elements($elem,"title",false));
$description = (get_elements($elem,"description",false));
```
/* do something with title and description */
...
/* get zero or more connections (axons) this element has ..*/
$axons = (get_elements($elem,"mesh:axon",true));

Simply put, you'll use the specific values you're looking for to find a specific item element. From there, you can pull the title and description. You can also pull the links to additional information, in the form of any axons.

Recall that, in both RDF and RSS, you have individual <item> elements in a channel, each with its <title> and <description>. Here you can see a major advantage of the RegExp-driven approach as opposed to DOM/SAX; you don't care about the positioning of <item> elements, either as child elements of the <channel>, or separate from it, since RegExps match text and are oblivious to the document structure!

Retrieving elements

Now that you've defined your database and created the parser functions, you're ready to have fun: Look up engagements, people and places in Springfield, as well as follow the associations, or links between them. Since you'll work with local feeds, you can use the following function to gather all the RSS content into a local file (see Listing 8).

Listing 8. Reading the RSS Database into memory

```php
function get_feed($url)
{
    /* open the feed locally */
    $fp = fopen ($url, "r");
    $content = fread($fp, filesize($url));
    return ($content);
}
```

And use it as shown in Listing 9.

Listing 9. Reading the RSS Database into memory (continued)

```php
$contacts_feed = get_feed("simpsons_contacts.xml");
$engagements_feed = get_feed("simpsons_engagements.xml");
$places_feed = get_feed("simpsons_places.xml");
```

Now, with the feeds in memory, let's proceed to actually look up stuff.
Classic lookups

Classic refers to the age old custom of looking up an element in a repository. Traditionally, this is with a SQL SELECT statement FROM some_table WHERE some condition is satisfied. No database is of any use if it cannot emulate the same functionality.

For example, what you might do with `SELECT * from ENGAGEMENTS` you do here with the code in Listing 10.

Listing 10. Emulating SELECT statements

```php
$engs = get_elements($engagements_feed, "item",true);
foreach ($engs as $eng)
{
    $title = (get_elements($eng,"title",false));
    echo ("Got Engagement: $title\n");
}
```

What if you want to get a list of all engagements, along with the participants? In SQL, you’d do some complex JOIN. Here, you simply follow your axons (see Listing 11).

Listing 11. Emulating a JOINed SELECT statement

```php
$engs = get_elements($engagements_feed, "item",true);
foreach ($engs as $eng)
{
    $title = (get_elements($eng,"title",false));
    echo ("Got Engagement: $title\n");
    $axons = (get_elements($eng,"mesh:axon",true));
    foreach ($axons as $ax)
    {
        if (strstr($ax, "Simpsons:/People")) {
            /* Lookup this person */
            $part = get_specific_element($contacts_feed,
               "guid",
               $ax); 
            if (!$part) { echo ($ax . " Not found\n");
            } else {
                $title = (get_elements($part,"title",false));
                echo ("Participant: $title\n");
            }
        }
    }
}
```
This sample is available at http://www.hisown.com/rss/lookuptest.php.

You accomplished your mission. You can do anything a table-based database can do. But can you improve on it?

Sample association

The last query you constructed wasn’t anything too novel; after all, any set of relational database tables can accomplish that. But to up the ante, you can now use a new type of query—an associative query, which shows similarities that items might have. Doing that in SQL and tables is not impossible, but can be quite excruciating. Doing that with your new associative format, however, is straightforward (see Listing 12).

Listing 12. An associative query

```php
$contacts_feed = get_feed("simpsons_contacts.xml");
$contacts = get_elements($contacts_feed,"item",true);
$common_axons = Array();
foreach ($contacts as $contact) {
    $title = get_elements($contact,"title",false);
    echo ("Got contact: $title<br/>
    $axons = (get_elements($contact,"mesh:axon",true));
    foreach ($axons as $ax) {
        /* Lookup these axons in the $common_axons */
        $cax_count = count ($common_axons);
        for($cax_no = 0; $cax_no < $cax_count; $cax_no++) {
            if ($common_axons[$cax_no] == $ax) {
                echo ("Bingo! Commonality found: $ax\n");
                exit(0);
            }
        }
        /* Add anyway */
        $common_axons [count($common_axons)] = $ax;
    } /* end foreach*/
}
```

And, its output, as expected, is shown in Listing 13.

Listing 13. Output

Got contact: Homer Jay Simpson
Got contact: Charles Montgomery_Burns
Bingo! Commonality found: Simpsons:/Places/Springfield_Nuclear_Power_Plant

This sample is attached, and available at http://www.hisown.com/rss/commontest.php
In Graph Theory terms, what has been done now is very similar to a Breadth First Search (BFS). This is really the tip of the iceberg; you only went one level deep, and your axons are unidirectional. True associations exploit depth and bidirectional links. You also ignored the link type, which you can use for specific searches, and more. This is incredibly useful in various areas of information processing and retrieval, as well as computational problems in graph theory, such as clustering. Another type of search you can try here is Depth First Search (follow associations recursively, child nodes, rather than sibling nodes). For more information, consider any Computer Science textbook on algorithms, specifically (see Resources for a link).

Sometimes, you want to perform only minimal parsing on the results, and leave them in XML form. This is useful, in particular, for Web services. You'll next see how to accomplish this at no extra cost; in fact, it's simpler to leave the results as they were. You can easily reassemble the XML fragments to form new valid documents.

Section 5. Putting together the end document

The data doesn't do any good if users can't use it. In this section, you'll look at returning raw XML data and using XSL to turn it into HTML.

Outputting Results as raw XML

As you saw, the RSS "database" will use the get_url() function, shown in Listing 8, to load the arbitrary RSS channels. The example just demonstrated parsed the elements from the RSS and displayed the specific details of interest. But you can just as easily gain advantage because the elements you isolate were, at one time, part of well formed XML. Therefore, you can embed them in your output, which itself will be XML. This enables you to make a custom XML document out of your feed aggregator, in any grammar you want, including RSS! This can allow a browser with RSS support or reader to automatically subscribe to the feed itself!

In your first example, the emulated join, you can slightly tweak it to output the elements themselves, rather than the participant titles. All it takes are three subtle modifications:

1. Instead of outputting the text ("Got Engagement ... " or "Participant..."), you echo the item itself: echo ("<item>...</item>

2. Declare this document to be an XML document (see Listing 14.)

Listing 14. Declaring the document to be an XML document

```
```
3. Envelop the elements in one container element and declare the namespaces (see Listing 15).

Listing 15. Enveloping the elements in one container element

```php
header("Content-Type: text/xml");
echo ('<?xml version="1.0"?>');
echo ('<rss xmlns:xics="blah"
xmlns:mesh="blahblah"
xmlns:xvcf="boo">');
echo ('<channel><title>Results</title>
...
echo ('</channel></rss> ');
```

To see this for yourself, go to http://hisown.com/rss/lookuptestxml.php. The output is well-formed XML, which you can leave as is for consumption by a Web service or RSS feed or choose to manipulate further, as is discussed next.

Presenting content to the userXSL

As shown above, your little RSS database is essentially a Web service, outputting its results in raw XML. You can then encapsulate this XML in XML/HTTP, or SOAP. The only problem is this XML isn't too human friendly. A human reader might care about the actual item titles, and descriptions (if available), but not about all the metadata of who/when/where it was published. Most readers do a pretty good job of hiding these details, and indeed just show the item titles and descriptions. One approach you took earlier was to filter the data to present only the fields of interest. That was easy enough, but at the cost of losing the benefit of the XML structure and metadata.

What if you want to customize the presentation layer, without violating your neat XML structure? Fortunately, XML has a solution for that—XML StyleSheet Language Transformations or XSLT. The author's previous tutorial, "Expand the editing capabilities of OpenOffice with XSLT" has already provided a "Crash course in XSLT" so rather than re-introduce to the technology here, I encourage you to check the Resources for a link.

Now let's create the actual stylesheet.

Creating your custom XML StyleSheet
If you know your XSLT (or have brushed up by consulting the "Crash course"), you know by now that XSLT syntax is somewhat complicated, and suffers from all the restrictions imposed by its being a subtype of XML (as in, it must be well-formed). Add to that the syntax imposed by the XSLT-specific grammar, restricted use of variables, and non-trivial constructs for flow control, and you achieve a degree of complication that encourages many to seek alternatives. The most common alternative is DOM processing in higher languages such as Java or JavaScript. Still, for your purposes here of merely rendering the presentation while leaving the actual output in raw XML form, XSL is perfect!

In the case here, you will use a simple, yet effective XSL template for your basic feed, to transform the feed into a human readable format.

Your XSL will look something like Listing 16.

**Listing 16. XSLT**

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:mesh="http://www.hisown.com/NameSpaces/Mesh/20071203"
xmlns:xvcf="http://www.hisown.com/NameSpaces/vcf/20071024">
<xsl:template match="/">
<HTML>
<BODY>
<xsl:for-each select="//item">
Item: <xsl:value-of select="./title" />
<br/>
<xsl:for-each select="./image">
<IMG title="{./title}" 
src="{./url}" width="{./width}" height="{./height}"/>
<br/>
</xsl:for-each>
</xsl:for-each>
</BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
```

To tie your XSL to the document, you insert an `<?xml-stylesheet?>` directive right after the document specifier (see Listing 17).

**Listing 17. Inserting an <?xml-stylesheet?> directive**

```php
..
echo ('<?xml version="1.0" ?>');
if ($_REQUEST['xsl'])
{
echo ('<?xml-stylesheet type="text/xsl" href="lookuptest.xsl" ?>');
}
```

And you're done! Try this with the same php, adding the argument: 

With the addition of the stylesheet to the XML output, the browser knew where to
find and retrieve it—that simple, that effective. Notice that, since your XSL resolved
the Image elements in the items and rendered them as HTML img elements, you
have a graphic display of your query results. A more complicated XSL can actually
draw a graph, or road map to show the interconnections between the elements and
highlight the associations.

Section 6. Summary

Next steps

This tutorial discussed RSS and demonstrated an unorthodox, yet innovative way to
use it in a relational database format. You use the core schema of RSS 2.0, but
extend it to store arbitrary items of any type. Each item is uniquely identified (by a
GUID). With a custom namespace, you can establish links between the items.

This is only a first step on a long road of many possibilities opened up by RSS. Do
you have Web site? Do you publish content? Consider using RSS, and joining the
millions that already do. With every Web site, blog or database that adopts RSS as
an alternative output mechanism, you come closer to the vision of the Semantic
Web, and of intelligent aggregators turned bots—that not only harvest and
concentrate information for the user by specific keywords, but actually take initiative
and perform actions by the very same!
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tutorial example code</td>
<td>x-rssdatabase-tut.zip</td>
<td>12KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Information about download methods
Resources

Learn

- **The RSS 1.0 specification**: Read about Atom, an XML-based Web content and metadata syndication format.

- **RSS 2.0 Specification**: Read more on this Web content syndication format and dialect of XML. All RSS files must conform to the XML 1.0 specification, as published on the World Wide Web Consortium (W3C) website.

- **RSS 2.0 and Atom**: Compare the differences between the RSS 2.0 and Atom 1.0 syndication languages.

- **Introduction to Syndication, (RSS) Really Simple Syndication** (Vincent Lauria, developerWorks, March 2006): Learn about RSS, Atom, and feed readers including why RSS so popular and what are its benefits? Learn what feed readers are available and which one might fit your needs.

- **RSS (file format)**: Read Wikipedia's excellent article detailing the history and differences of RSS file formats.

- **The future of the Web is Semantic** (Naveen Balani, developerWorks, October 2005): Explore the basics of Semantic Web technologies and how you can leverage ontology-based development.

- **Expand the editing capabilities of OpenOffice with XSLT** (Jonathan Levin, developerWorks, October 2007): In this tutorial, learn how to use OpenOffice's import/export filters to open your XML data as though it's just a plain document.

- **XSLT: Working with XML and HTML** (Khun Yee Fung, Addison-Wesley, December 2000): Try a comprehensive reference and tutorial to XSLT.

- **XSLT Functions**: Check out the extensive reference from the w3school.com.

- **IBM XML certification**: Find out how you can become an IBM-Certified Developer in XML and related technologies.

- **XML technical library**: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.

- **developerWorks technical events and webcasts**: Stay current with technology in these sessions.

- **developerWorks XML zone**: Learn all about XML.

- **The technology bookstore**: Browse for books on these and other technical topics.

Get products and technologies

- **OpenOffice.org**: Download the product and find all sorts of information about it.
• **IBM trial software**: Build your next development project with trial software available for download directly from developerWorks.

**Discuss**

• **XML zone discussion forums**: Participate in any of several XML-related discussions, including the **Atom and RSS forum**.

• **developerWorks XML zone**: Share your thoughts: After you read this article, post your comments and thoughts in this forum. The XML zone editors moderate the forum and welcome your input.

• **developerWorks blogs**: Check out these blogs and get involved in the developerWorks community.

**About the author**

Jonathan Levin

Jonathan Levin is a freelance trainer and consultant in a myriad array of topics, ranging from XML/XSLT and Ajax, through low-level protocols and device drivers for Linux and Windows. He has designed and developed a lightweight Web-desktop environment called JADE (http://jade.hisown.com) that provides enterprise-class email, calendaring, RSS, and full remote file system functionality which makes extensive use of XML and XSLT along with PHP in the backend and JavaScript in the frontend.

**Trademarks**

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. UNIX is a registered trademark of The Open Group in the United States and other countries. Other company, product, or service names may be trademarks or service marks of others.