

# Explore practical data mining and parsing with PHP

## Dig into XML and HTML data to find useful information with PHP

Skill Level: Introductory

[Eli White](#)  
CTO  
MojoLive

26 Jul 2011

The art of data mining is a wide field, and mentioning the term to two different developers gives you two very different ideas about it. In this article, you learn what data mining is, its importance, different ways to accomplish data mining (or to create web-based data mining tools) and develop an understanding of XML structure to parse XML and other data in PHP technology.

## Data mining and its importance

### Frequently used acronyms

- API: Application programming interface
- CDATA: Character data
- DOM: Document Object Mode
- FTP: File Transfer Protocol
- HTML: HyperText Markup Language
- HTTP: Hypertext Transfer Protocol
- REST: Representational State Transfer
- URL: Uniform Resource Locator

- W3C: World Wide Web Consortium
- XML: Extensible Markup Language

Wikipedia defines data mining as "the process of extracting patterns from large data sets by combining methods from statistics and artificial intelligence with database management." This is a very deep definition and probably goes beyond the typical use case for most people. Few people work with artificial intelligence; most commonly, data mining simply entails the ingesting of large data sets and searching through them to find information that is useful.

Given how the Internet has grown, with so much information available, it is important to be able to aggregate large amounts of data and make some sense of it. To take datasets much larger than a single person can read and boil them down to useful data is a primary goal. This type of data mining is the focus of this article, specifically how to collect and parse this data.

## Practical uses of data mining

Data mining has many practical uses. You might want to scour a website for information that it provides (such as attendance records for movies or concerts). You might have more serious information, such as voter records, to retrieve and make sense of the data. Or, more commonly, you might look at social network data and attempt to parse trends from it, such as how often your company is mentioned and whether it's mentioned in a positive or negative light.

## Precautions before mining a website

Before you continue, I should mention that I assume you will pull this data from another website. If you already have the data at your disposal, that's a very different situation. When you pull data from a website, you need to make sure that you are following the terms of service regardless of whether you are web scraping (more on this later) or using an API. If you are scraping, you also need to be wary of following the site's robots.txt file, which describes what parts of the website scripts you can access. Finally, make sure that you are respectful of the site's bandwidth. You should not write scripts that access the site's data as fast as your script can run. Not only might you cause hosting problems, but you run the risk of being banned or blocked from the site for being too aggressive.

## Understanding XML data structure

Regardless of the way that you pull data in, chances are that you will receive data in

XML (or HTML) format. XML has become the standard language of the Internet when it comes to sharing data. It's important to briefly consider XML structure and how to handle it in PHP before you look at methods to retrieve it.

The basic structure of an XML document is very straightforward, especially if you have previously worked with HTML. All data in an XML document is stored in one of two ways. The primary way to store the data is inside nested tags. For an example of the simplest form, suppose that you have an address, which can be stored in a document such as this:

```
<address>1234 Main Street, Baltimore, MD</address>
```

You can nest these XML data points to create a list of multiple addresses. You can put all of these addresses inside another tag, in this case called `locations` (see [Listing 1](#)).

### Listing 1. Multiple addresses in XML

```
<locations>
  <address>1234 Main Street, Baltimore, MD</address>
  <address>567 1st Street, San Jose, CA</address>
  <address>901 Washington Ave, Chicago, IL</address>
</locations>
```

To expand this approach further, you might want to break the addresses into their constituent parts of street, city, and state, which makes processing of the data easier. In that case, you have a more typical XML file, as in [Listing 2](#).

### Listing 2. Fully broken-down addresses in XML

```
<locations>
  <address>
    <street>1234 Main Street</street>
    <city>Baltimore</city>
    <state>MD</state>
  </address>
  <address>
    <street>567 1st Street</street>
    <city>San Jose</city>
    <state>CA</state>
  </address>
  <address>
    <street>901 Washington Ave</street>
    <city>Chicago</city>
    <state>IL</state>
  </address>
</locations>
```

As mentioned, you can store XML data in two main ways. You've now seen one of them. The other method is through attributes. Each tag can have a number of

attributes assigned to it. While less common, this approach can be a very useful tool. Sometimes it gives additional information, such as a unique ID or an event date. Quite often, it adds meta data; in your address example, a `type` attribute indicates whether the address is a home or work address, as in [Listing 3](#).

### Listing 3. Tags added to XML

```
<locations>
  <address type="home">
    <street>1234 Main Street</street>
    <city>Baltimore</city>
    <state>MD</state>
  </address>
  <address type="work">
    <street>567 1st Street</street>
    <city>San Jose</city>
    <state>CA</state>
  </address>
  <address type="work">
    <street>901 Washington Ave</street>
    <city>Chicago</city>
    <state>IL</state>
  </address>
</locations>
```

Note that XML documents do always have a parent root tag/node that all other tags/nodes are children of. XML also can include other declarations and definitions at the beginning of the document and a few other complications (such as CDATA blocks). I highly recommend that you read more about XML in [Resources](#).

## Parsing XML data in PHP

Now that you understand what XML looks like and how it's structured, you need to know how to parse and programmatically access that data inside PHP. A number of libraries created for PHP allow XML parsing, and each library has its own benefits and drawbacks. There are DOM, XMLReader/Writer, XML Parser, SimpleXML, and others. For the purposes of this article, I focus on SimpleXML as it is one of the most commonly used libraries and one of my favorites.

SimpleXML, as its name suggests, was created to provide a very simple interface to accessing XML. It takes an XML document and transforms it into an internal PHP object format. Accessing data points becomes as easy as accessing object variables. Parsing an XML document with SimpleXML is as easy as using the `simplexml_load_file()` function (see [Listing 4](#)).

### Listing 4. Parsing a document with SimpleXML

```
<?php
$xml = simplexml_load_file('listing4.xml');
?>
```

That's really all that is required. Do note that thanks to PHP's filestream integration, you can insert a filename or a URL here and the filestream integration automatically fetches it. You can also use `simplexml_load_string()` if you already have the XML loaded into memory. If you run this code on the XML in [Listing 3](#) and use `print_r()` to see the rough structure of the data, you get the output in [Listing 5](#).

### Listing 5. Output of parsed XML

```
SimpleXMLElement Object
(
    [address] => Array
        (
            [0] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [type] => home
                        )
                    [street] => 1234 Main Street
                    [city] => Baltimore
                    [state] => MD
                )
            [1] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [type] => work
                        )
                    [street] => 567 1st Street
                    [city] => San Jose
                    [state] => CA
                )
            [2] => SimpleXMLElement Object
                (
                    [@attributes] => Array
                        (
                            [type] => work
                        )
                    [street] => 901 Washington Ave
                    [city] => Chicago
                    [state] => IL
                )
        )
    )
)
```

You can then access the data using standard PHP object access and methods. For example, to echo out every state that someone lived in, you can iterate over the addresses to do just that (see [Listing 6](#)).

### Listing 6. Iterating over addresses

```
<?php
$xml = simplexml_load_file('listing4.xml');

foreach ($xml->address as $address) {
    echo $address->state, "<br \>\n";
}
```

```
?>
```

Accessing the attributes is a little different. Rather than reference them as you do an object property, you access them like array values. You can change that last code sample to show the `type` attribute by using the code in [Listing 7](#).

### Listing 7. Adding attributes

```
<?php
$xml = simplexml_load_file('listing4.xml');

foreach ($xml->address as $address) {
    echo $address->state, ': ', $address['type'], "<br \>\n";
}
?>
```

While all the current examples involved iteration, you can reach directly into the data and use a specific piece of information that you want, such as grabbing the street address of the second address with the code `$xml->address[1]->street`.

You should now have the basic tools to start playing with XML data. I do recommend that you read the SimpleXML documentation and other links listed in [Resources](#) to learn more.

## Data mining in PHP: Possible ways

As mentioned, you can access data in multiple ways. The two primary methods are web scraping and API use.

### Web scraping

Web scraping is the act of literally downloading entire web pages programmatically and extracting data from the page. There are entire books written on this subject (see [Resources](#)). I briefly list the tools needed to do this. First of all, PHP makes it very easy to read a web page in as a string. There are many ways to do this, including using `file_get_contents()` with a URL, but in this case you want to be able to parse the HTML in a meaningful manner.

Given that HTML is at its heart a language based on XML, it is useful to convert HTML into a SimpleXML structure. You can't just load an HTML page using `simplexml_load_file()`, however, as even valid HTML isn't XML. A good workaround is to use the DOM extension to load the HTML page as a DOM document and then convert it to SimpleXML, as in [Listing 8](#).

### Listing 8. Using DOM methods to get a SimpleXML version of a web page

```
<?php
$dom = new DOMDocument();
$dom->loadHTMLFile('http://example.com/');
$xml = simplexml_import_dom($dom);
?>
```

After you've done this, you can now traverse the HTML page just as you might have any other XML document. Therefore you can access the title of the page now using `$xml->head->title` or go deep into the page with references such as `$xml->body->div[0]->div[0]->div[0]->h4[0]`.

As you might expect from that last example, though, it can get very unwieldy at times to try to find data in the midst of an HTML page, which often isn't nearly as organized as an XML file is. The above line looks for the first h4 that exists inside of three nested divs; in each case, it looks for the first div inside each parent.

Luckily, if you want to find only the first h4 on the page, or other such "direct data," XPath is a much easier way to do so. XPath is essentially a way to search through XML documents using a query language, and SimpleXML exposes this. XPath is a very powerful tool and can be the subject of an entire series of articles, including some listed in [Resources](#). In basic terms, you use ' / ' to describe hierarchical relationships; therefore, you can rewrite the preceding references as the following XPath search (see [Listing 9](#)).

### Listing 9. Using XPath directly

```
<?php
$h4 = $xml->xpath('/html/body/div/div/div/h4');
?>
```

Or you could just use the ' // ' option with XPath, which causes it to search all of the document for the tags you are looking for. Therefore, you could find all the h4's as an array, then access the first one, using XPath:

```
'//h4'
```

### Walking an HTML hierarchy

The main reason to talk about these conversions and XPath is that one of the common required tasks when you do web scraping is to automatically find other links on the web page and follow them, allowing you to "walk" the website, finding out as much information as possible.

This task is made fairly trivial using XPath. [Listing 10](#) gives you an array of all the <a> links with "href" attributes, allowing you to handle them.

### Listing 10. Combining techniques to find all links on a page

```
<?php
$dom = new DOMDocument();
$dom->loadHTMLFile('http://example.com/');
$xml = simplexml_import_dom($dom);
$links = $xml->xpath('//a[@href]');
foreach ($links as $l) {
    echo $l['href'], "<br />\n";
}
?>
```

Now that code finds all the `<a href="">` links, but you might quickly start crawling the entire web if you followed every possible link you found. Therefore, it's best to enhance your code to ensure that you access only links that are valid HTML links (not FTP or JavaScript) and that go back only to the same website (either through full domain links or through relative ones).

An easier way is to iterate on the links using PHP's built-in `parse_url()` function, which handles a lot of the sanity checks for you. [Listing 11](#) looks something like this.

### Listing 11. A more robust site walker

```
<?php
$dom = new DOMDocument();
$host = 'example.com';
$dom->loadHTMLFile("http://{ $host }/");
$xml = simplexml_import_dom($dom);
$links = $xml->xpath('//a[@href]');
foreach ($links as $l) {
    $p = parse_url($l['href']);
    if (empty($p['scheme']) || in_array($p['scheme'], array('http', 'https'))) {
        if (empty($p['host']) || ($host == $p['host'])) {
            echo $l['href'], "<br />\n"; // Handle URL iteration here
        }
    }
}
?>
```

As a final note on HTML parsing, you reviewed how to use the DOM extension solely for the purpose of converting back into SimpleXML, for a unified interface to all XML-like languages. Note that the DOM library itself is a very robust one and can be used directly. If you are well versed in JavaScript and traversing a DOM document tree using tools such as `getElementsByTagName`, then you might be comfortable staying within the DOM library and not using SimpleXML.

You should now have the tools that you need to start scraping data from web pages. Once you are familiar with the techniques detailed previously in this article, you can read any information from the web page, not just the links that you can follow. We hope that you don't need to do this task because an API or other data source exists instead.

### Using XML APIs and data



At this point, you have the basic skills to access and use a majority of the XML data APIs on the Internet. They are often REST-based and therefore require only a simple HTTP access to retrieve the data and parse it using the preceding techniques.

Every API is different in the end. You certainly can't cover how to access every single one so let's walk through some basic examples of XML APIs. One of the most common sources of data, and already in XML format, is the RSS feed. RSS stands for Really Simple Syndication and is a mostly standardized format for sharing frequently updated data, such as blog posts, news headlines, or podcasts. To learn more about the RSS format, see [Resources](#). Note that RSS is an XML file, with a parent tag called <channel> that can have any number of <item> tags in it, each providing a bevy of data points.

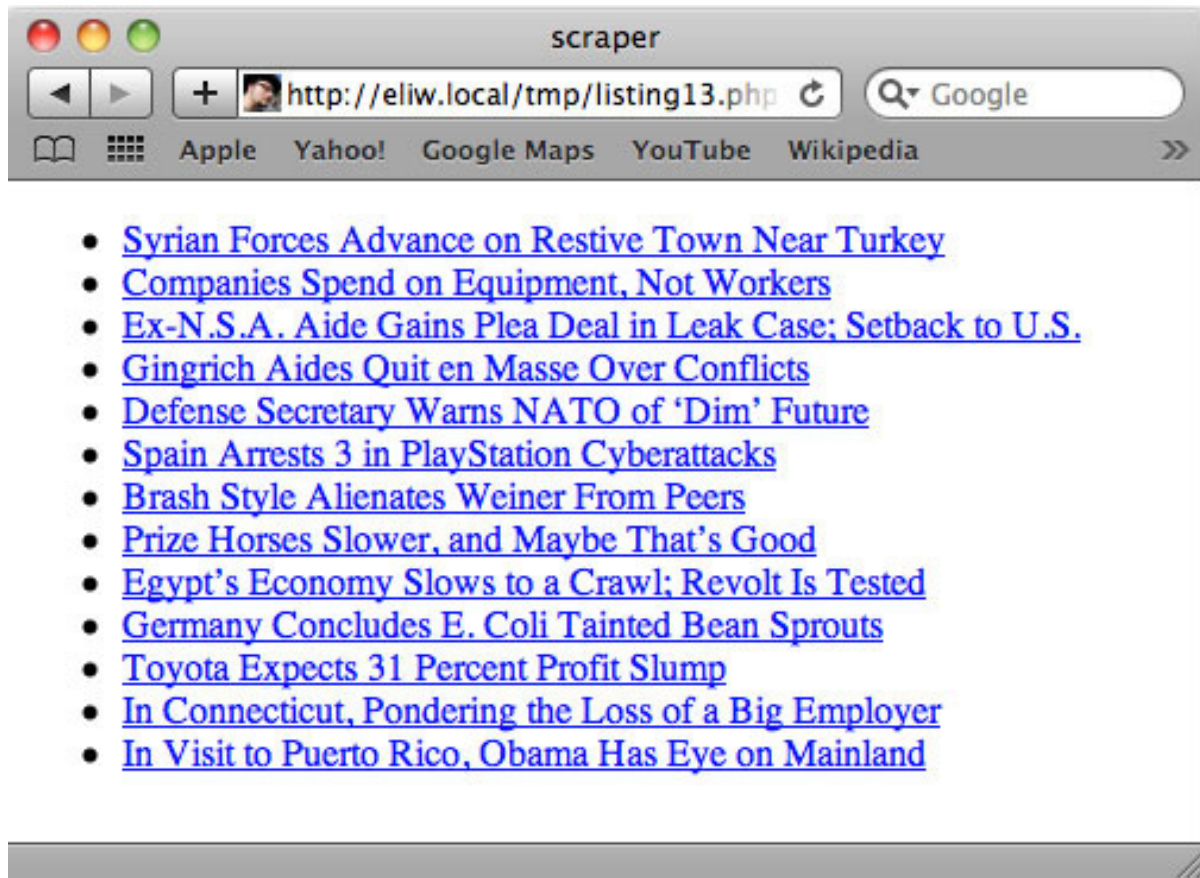
As an example, use SimpleXML to read in the RSS feed of the headlines of The New York Times (see [Resources](#) for a link to the RSS feed) and format a list of headlines with links to the stories (see [Listing 12](#)).

### Listing 12. Reading The New York Times RSS feed

```
<ul>
<?php
$xml = simplexml_load_file('http://www.washingtonpost.com/rss/homepage');
foreach ($xml->channel->item as $item) {
    echo "<li><a href=\"{".$item->link}\">{".$item->title}
</a></li>";
}
?>
</ul>
```

[Figure 1](#) shows the output from The New York Times feed.

### Output from The New York Times feed



Now, let's explore an example of a more fully featured REST-based API. A good one to start with is the Flickr API because it offers lots of data without the need to authenticate with it. Many APIs require you to authenticate with them, using OAuth or other mechanisms, to act on behalf of a web user. This step might apply to the entire API, or just part of it. Check the documentation of each API for how to do this.

To demonstrate using the Flickr API for a non-authenticated request, you can use its search API. For an example, search Flickr for all public photos of crossbows. While you don't need to authenticate, as you might with many APIs, you do need to generate an API key to use when accessing the data. Learn to do that task directly from Flickr's API documentation itself. After you have an API key, you can explore using their search feature as in [Listing 13](#).

### Listing 13. Searching for "crossbow" using the Flickr API

```
<?php
// Store some basic information that you need to reference
$apiurl = 'http://api.flickr.com/services/rest/?';
$key = '9f275087e222ee395c92662437bf84a2'; // Replace with your own key

// Build an array of parameters that you want to request:
$params = array(
    'method' => 'flickr.photos.search',
```

```

    'api_key' => $key,
    'text' => 'crossbow', // Our search term
    'media' => 'photos',
    'per_page' => 20 // We only want 20 results
);

// Now make the request to Flickr:
$xml = simplexml_load_file($apiurl . http_build_query($params));

// From this, iterate over the list of photos & request more info:
foreach ($xml->photos->photo as $photo) {
    // Build a new request with this photo's ID
    $params = array(
        'method' => 'flickr.photos.getInfo',
        'api_key' => $key,
        'photo_id' => (string)$photo['id']
    );
    $info = simplexml_load_file($apiurl . http_build_query($params));

    // Now $info holds a vast amount of data about the image including
    // owner, GPS, dates, description, tags, etc ... all to be used.

    // Let's also request "sizes" to get all of the image URLs:
    $params = array(
        'method' => 'flickr.photos.getSizes',
        'api_key' => $key,
        'photo_id' => (string)$photo['id']
    );
    $sizes = simplexml_load_file($apiurl . http_build_query($params));
    $small = $sizes->xpath("//size[@label='Small']");

    // For now, just going to create a simple display of the image,
    // linked back to Flickr, with title, GPS info, and more shown:
    echo <<<EOHTML
<div>
  <a href="{ $info->photo->urls->url[0] }">
    
  </a>
  <ul>
    <li>Title: { $info->photo->title }</li>
    <li>User: { $info->photo->owner['realname'] }</li>
    <li>Date Taken: { $info->photo->dates['taken'] }</li>
    <li>Location: { $info->photo->location->locality },
      { $info->photo->location->county },
      { $info->photo->location->region },
      { $info->photo->location->country }
    </li>
  </ul>
</div>
EOHTML;
}
?>

```


**Figure 2** shows the output of the Flickr program. The results of your search for crossbows includes photos plus information about each photo (title, user, location, date the photo was taken).

### **Figure 2. Example output of the Flickr program from Listing 13**


scraper

http://eliw.local/tmp/scraper.php


Apple Yahoo! Google Maps YouTube Wikipedia News (1005)



- o Title: Seagirt 091
- o User: (JacquelineLefleur)
- o Date Taken: 2010-06-11 15:38:55



- o Title: Roman Pharos and Saxon Church from the Great Tower of Dover Castle, Kent, UK
- o User: John Latter (John Latter)
- o Date Taken: 2011-05-20 17:12:02
- o Location: Dover, Kent, England, United Kingdom



You can see how powerful APIs like this are and how you can combine various calls in the same API to get the data that you need. With these basic techniques, you can mine the data of any website or information source.

Simply discover how you can get programmatic access to the data through an API or web scraping. Then use the methods shown to access and iterate over all the target data.

## Storing and reporting on extracted data

The final point, storing and reporting on the data, is in many ways the easiest part—and perhaps the most fun. The sky is the limit here as you decide how to handle this aspect for your own situation.

Typically, take all of the information that you gather and store it in a database. Then structure the data in a way that matches how you plan to access it later. When doing this, don't be shy about storing more information than you think you might need. While you can always delete data, retrieving additional information can be a painful process once you have lots of it. It's better to overestimate in the beginning. After all, you never know what piece of data might turn out to be interesting.

Then at that point, after the data is stored in a database or similar data store, you can create reports. Reporting might be as simple as running some basic SQL queries against a database to see the number of times that a piece of data exists, or it might be very complicated web user interfaces designed to let someone dive in and find their own correlations.

After you do the hard work of cataloging all the data, you can imagine creative ways to display it.

## Conclusion

Through the course of this article, you looked at the basic structure of XML documents and an easy method to parse those in PHP using SimpleXML. You also added the ability to handle HTML in a similar manner and touched on the basics of walking a website to scrape data not available in an XML format. Using these tools, and following some of the examples that have been given, you now have a good base level of knowledge so you can begin to work on data mining a website. There is much more to learn than a single article can convey. For additional ways to increase your knowledge about data mining, plan to check the [Resources](#).

## Downloads

Description	Name	Size	Download method
source code	datamining_source.zip	10KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- [XML](#) as described on Wikipedia: Read a description of the XML specification.
- [Extensible Markup Language \(XML\) 1.0 \(Fifth Edition\)](#) (W3C Recommendation, November 2008): Visit this source for specific details about XML features.
- [Introduction to XML](#) (Doug Tidwell, developerWorks, August 2002): Look at what XML is, why it was developed, and how it shapes electronic commerce. Review a variety of important XML programming interfaces and standards, and two case studies of how companies solve business problems with XML.
- [XML Tutorial](#) (W3Schools): Read a lesson about XML and how it can transport and store data.
- [SimpleXML documentation](#): Browse and learn about a tool set to convert XML to an object that you can process with normal PHP property selectors and array iterators.
- [php|architect's Guide to Web Scraping with PHP](#) (Matthew Turland): Get more information on web scraping with a variety of technologies and frameworks.
- [XML Path Language \(XPath\) Version 1.0](#) (W3C Recommendation, November 1999): Familiarize yourself with the specification for a common syntax and semantics for functionality shared between XSLT and XPointer.
- [Get started with XPath](#) (Bertrand Portier, developerWorks, May 2004): Cover the basics of the XML Path Language, or XPath.
- [XPath Tutorial](#) (W3Schools): Read a lesson about XPath and how to navigate through elements and attributes in an XML document.
- [RSS specification](#): Explore the details of the RSS web content syndication format.
- [Flickr Services](#): Look into the Flickr API, an online photo management and sharing application.
- [How to use regular expressions in PHP](#) (Nathan A. Good, developerWorks, January 2006): Learn to validate user input, parse user input and file contents, and reformat strings.
- [PHP.net](#): Visit and explore the central resource for PHP developers.
- [Recommended PHP reading list](#) (Daniel Krook and Carlos Hoyos, developerWorks, March 2006): Learn about PHP (Hypertext Preprocessor) with this reading list compiled for programmers and administrators by IBM web application developers.

- [PHP and more](#): Browse all the PHP content on developerWorks.
- [Zend Core for IBM](#): Using a database with PHP? Check out a seamless, out-of-the-box, easy-to-install PHP development and production environment that supports IBM DB2 V9.
- [RSS feed of the headlines of The New York Times](#): Experiment with an RSS feed of headlines from The New York Times.
- [New to XML?](#) Get the resources you need to learn XML.
- [XML area on developerWorks](#): Find the resources you need to advance your skills in the XML arena. See the [XML technical library](#) for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks
- Expand your PHP skills by checking out the IBM developerWorks [PHP project resources](#).
- [IBM XML certification](#): Find out how you can become an IBM-Certified Developer in XML and related technologies.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- [developerWorks on Twitter](#): Join today to follow developerWorks tweets.
- [developerWorks podcasts](#): Listen to interesting interviews and discussions for software developers.
- [developerWorks on-demand demos](#): Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.

## Get products and technologies

- [PHP](#): Get this general-purpose scripting language for web development.
- [IBM product evaluation versions](#): Download or [explore the online trials in the IBM SOA Sandbox](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

## Discuss

- [PHP Forum: Developing PHP applications with IBM Information Management products \(DB2, IDS\)](#): Post questions and share insights on building better web apps with PHP, IBM Information Management products (such as DB2, Informix, and others), the XML capabilities (XML, XSLT, SQL/XML, XQuery) of IBM data servers in this developerWorks forum.
- [XML zone discussion forums](#): Participate in any of several XML-related discussions.



- The [developerWorks community](#): Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

## About the author

Eli White



Eli White is currently CTO of MojoLive and has worked on creating web applications for over 16 years, with the last 10 spent exclusively on PHP projects. He has worked a number of varied jobs in the past, including jobs at Zend, TripAdvisor, Digg, and the Hubble Space Telescope Program. He is the author of "PHP 5 in Practice" and regularly presents at conferences.

## Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of [IBM trademarks](#).

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Other company, product, or service names may be trademarks or service marks of others.