Using PHP With LDAP (part 1)

By Harish Kamath
2003-04-04

Printed from DevShed.com
URL: http://www.devshed.com/Server_Side/PHP/PHPwithLDAP1/

Plugging In

One of the reasons for PHP's immense popularity is its support for a wide variety of different Internet technologies. It is this support, coupled with the speed with which new language extensions get added to the source tree, that keeps PHP ahead of other competing languages in the same category, and ensures the continuity (and future growth) of the fan following it currently enjoys.

One such Internet technology is LDAP, the Lightweight Directory Access Protocol. In case you're not familiar with LDAP, it is a protocol designed to allow quick, efficient searches of directory services. Built around Internet technologies, LDAP makes it possible to easily update and query directory services over standard TCP/IP connections, and includes a host of powerful features, including security, access control, data replication and support for Unicode.

PHP has shipped with support for LDAP since version 3.x of the language, and today comes with built-in support to connect to LDAP databases, perform queries and process result sets. These capabilities make PHP extremely popular among developers who need to create Web applications that interface with LDAP directories - for example, online address books and Web-based directories and directory search agents. Though these sound like intimidating projects to take on, they're actually pretty simple - as you'll see, PHP makes otherwise complex processes seem almost friendly via its powerful functions and sophisticated capabilities.

Over the next few pages, I'm going to introduce you to the LDAP functions that ship with PHP, demonstrating how they can be used to rapidly create a Web-based interface to an LDAP directory; this interface will allow users to perform queries and add and remove entries from the directory using a standard Web browser. I'll assume here that you have a working knowledge of PHP fundamentals, a development environment with a working PHP build, and a sense of humour. LDAP knowledge would be a bonus, although it is not a requirement - this tutorial does include a brief discussion of LDAP basics.

Got 'em all? Flip the page, and let's get going!

Looking For Answers

In case you're not familiar with LDAP, this section is designed to give you a crash course in the basics. It is not intended to be exhaustive - you should take a look at the links at the end of this article for more detailed material - but it should get you through the remainder of this article without confusing you too much.

An LDAP directory is usually structured hierarchically, as a tree of nodes (the LDAP directory tree is sometimes referred to as the Directory Information Tree, or DIT). Each node represents a record, or "entry", in the LDAP database.

An LDAP entry consists of numerous attribute-value pairs, and is uniquely identified by what is known as a "distinguished name" or "DN". If you're familiar with RDBMS, it's pretty easy to draw an analogy here: an LDAP entry is analogous to a record, its attributes are the fields of that record, and a DN is a primary key that uniquely identifies each record.

Consider the following example of an LDAP entry, which might help make things clearer:

```
dn: mail=sue@my-domain.com, dc=my-domain, dc=com
objectclass: inetOrgPerson
cn: Sue
sn: Jones
mail: sue@my-domain.com
telephoneNumber: 1 234 567 8912
```

This is an entry for a single person, Sue Jones. As you can see, the different components of the entry - name, email address, telephone number - are split into attribute-value pairs, with the entire record identified by a unique DN (the first line of the entry). Some of these attributes are required and some are optional, depending on the object class being used for the entry (more on this later); however, the entire set of data constitutes a single entry, or node, on the LDAP directory tree.

Since LDAP entries are arranged in a hierarchical tree, and since each node on the tree can be uniquely identified by a DN, the LDAP model lends itself well to sophisticated queries and powerful search filters. For example, I could restrict my search to a particular subset of the tree simply by specifying a different base for the query to begin from, or query only against specific attributes in the directory tree. Heck, I could even do both, and feel like a Real Programmer!

The Bare Necessities
In order to get started with PHP and LDAP, there are a couple of things you need.

First, you need a LDAP server, which you can use as the foundation for your PHP scripts. While there are many commercial LDAP servers, the one used in this article is the OpenLDAP server, which is robust, scalable and - best of all - free. You can download a copy from http://www.openldap.org(this tutorial uses OpenLDAP 2.1.16).

Once you've got your LDAP server up and running, it's time to populate it with some entries. Here are the sample entries that I will be using throughout this tutorial.

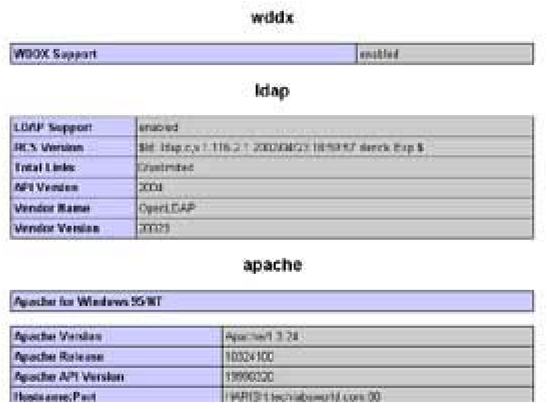
```
dn: dc=my-domain, dc=com
objectClass: dcObject
objectClass: organization
o: MyOrganization
dc: my-domain.com

dn: mail=root@my-domain.com, dc=my-domain, dc=com
objectClass: inetOrgPerson
cn: Keith
sn: Richards
mail: root@my-domain.com

dn: mail=joe@my-domain.com, dc=my-domain, dc=com
objectClass: inetOrgPerson
cn: Joe
sn: Somebody
mail: joe@my-domain.com

dn: mail=sarah@my-domain.com, dc=my-domain, dc=com
objectClass: inetOrgPerson
```

```
cn: Sarah
sn: Nobody
mail: sarah@my-domain.com
telephoneNumber: 23 67 128 5639
```

Refer to the OpenLDAP reference manual, or the usage guide that shipped with your LDAP server, for the exact commands needed to import this data into your LDAP directory. Note that the server must be set up to allow read access without a password, and its base DN should be "dc=my-domain, dc=com", as in the sample data above.

Once the LDAP server has been set up, the next step is to configure PHP to communicate with this server. PHP 4.3 supports this via its LDAP extension, which is not activated by default - you may need to recompile your PHP binary with the "--with-ldap" parameter to activate LDAP support (Windows users get a pre-built binary with their distribution).

Once you've got PHP configured with LDAP support, check to see that all is well via a quick call to phpinfo() - you should see something like this:



wddx

WDDX Support	enabled

ldap

LDAP Support	enabled
RCS Version	$Id: ldap.c,v 1.116.2.1 2003/04/23 18:59:57 derick Exp $
Total Links	Unlimited
API Version	2004
Vendor Name	OpenLDAP
Vendor Version	20023

apache

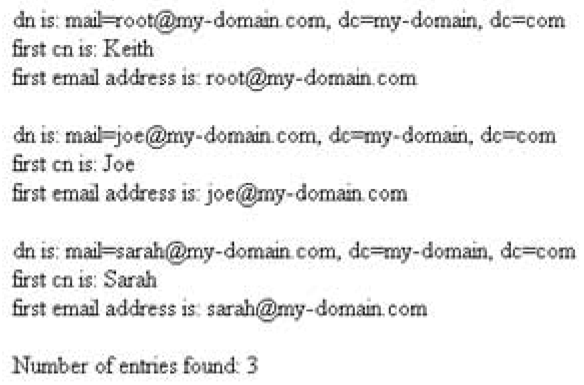
Apache for Windows 95/NT

Apache Version	Apache/1.3.24
Apache Release	10324100
Apache API Version	19990320
Hostname:Port	[illegible]

Code Poet
Now that all the requirements are in place, let's put together a simple PHP script to connect to the LDAP server and display the contents of the directory. Take a look at the following code.

```
<html>
<head>
</head>

<body>

<?php

// specify the LDAP server to connect to
$conn = ldap_connect("localhost") or die("Could not connect to
server");

// bind to the LDAP server specified above
$r = ldap_bind($conn) or die("Could not bind to server");

// start searching
// specify both the start location and the search criteria
// in this case, start at the top and return all entries $result =
ldap_search($conn,"dc=my-domain,dc=com", "(cn=*)") or die ("Error in
search
query");

// get entry data as array
$info = ldap_get_entries($conn, $result);

// iterate over array and print data for each entry
for ($i=0; $i<$info["count"]; $i++)
{
    echo "dn is: ". $info[$i]["dn"] ."<br>";
    echo "first cn is: ". $info[$i]["cn"][0] ."<br>";
    echo "first email address is: ". $info[$i]["mail"][0] ."<p>"; }

// print number of entries found
echo "Number of entries found: " . ldap_count_entries($conn, $result) .
"<p>";

// all done? clean up
ldap_close($conn);

?>

</body>
</html>
```

Run the script in your browser, and you should see something like this:



```
dn is: mail=root@my-domain.com, dc=my-domain, dc=com
first cn is: Keith
first email address is: root@my-domain.com

dn is: mail=joe@my-domain.com, dc=my-domain, dc=com
first cn is: Joe
first email address is: joe@my-domain.com

dn is: mail=sarah@my-domain.com, dc=my-domain, dc=com
first cn is: Sarah
first email address is: sarah@my-domain.com

Number of entries found: 3
```

Let's take a closer look at this script.

Anatomy 101
At the onset, I should tell you that communicating with an LDAP server through PHP is very similar to communicating with a database server - open a connection, use the connection to search for some information, iterate over the resource handle containing the results of the search and close the connection.

1. Let's begin with the basic connection.

```
<?php

// specify the LDAP server to connect to
$conn = ldap_connect("localhost") or die("Could not connect to
server");
```

```
?>
```

The ldap_connect() function is used to open a connection with the LDAP server. If a connection is possible, the function returns a link identifier that is used for all subsequent communication with the LDAP server. If a connection is not possible, the function returns false.

By default, the ldap_connect() function looks for an LDAP server on port 389. In case your LDAP server is running on a non-standard port, you can specify that port number as an additional argument to ldap_connect(), as in the following code snippet.

```
<?php

// LDAP server running on non-standard port 510
$conn = ldap_connect("localhost", 510) or die("Could not connect to
server");

?>
```

2. Once a connection is established, the next step is to "bind" it to the LDAP server via the ldap_bind() function call.

```
<?php

// bind to the LDAP server specified above
$r = ldap_bind($conn) or die("Could not bind to server");

?>
```

In the example above, this is a so-called "anonymous" bind, as no authentication credentials are provided in my call to ldap_bind().

3. Now for the meat of the script - the ldap_search() function.

```
<?php

// start searching
// specify both the start location and the search criteria
// in this case, start at the top and return all entries $result =
ldap_search($conn,"dc=my-domain,dc=com", "(cn=*)") or die ("Error in
search
query");

?>
```

This function requires three parameters - the LDAP link identifier, the DN of the location in the LDAP hierarchy where the search should begin, and the search query string itself. In the example above, I'm asking the system to begin searching at the root of the hierarchy, and return all entries which have a valid common name ("cn") attribute.

The return value of ldap_search() is a result set with the entries that match the query.

4. Once a result set has been obtained, the entries within it may be placed in a multi-dimensional array via a call to ldap_get_entries().

```
<?php
```

```
// get entry data as array
$info = ldap_get_entries($conn, $result);

// iterate over array and print data for each entry
for ($i=0; $i<$info["count"]; $i++)
{
    echo "dn is: ". $info[$i]["dn"] ."<br>";
    echo "first cn is: ". $info[$i]["cn"][0] ."<br>";
    echo "first email address is: ". $info[$i]["mail"][0] ."<p>"; }

?>
```

Every element of this array corresponds to one entry from the LDAP directory. Each of these elements (entries) is further structured as an array, whose elements correspond to the attributes of each entry; these attributes may be accessed either by name or index number. A "for" loop is used to iterate over this multi-dimensional array, and return a subset of the data within it.

A count of all the entries in the result set can be obtained via a call to ldap_count_entries()

```
<?php

// print number of entries found
echo "Number of entries found: " . ldap_count_entries($conn, $result) .
"<p>";

?>
```

5. Once the result set has been processed, the connection to the LDAP server can be closed via a call to ldap_close().

```
<?php

// all done? clean up
ldap_close($conn);

?>
```

It's generally a good idea to close the connection once you're done using it, so that system resources used by the script are freed up and made available to other programs.

The five steps outlined above make up a general template for interacting with an LDAP server through PHP. You will see that these steps (and the code that they consist of) recur over almost every example in this tutorial.

What's In A Name?
The example on the previous page was fairly static - all it did was return a list of all the entries in the directory that had a "cn" attribute. This next example makes things a little more interactive - it includes a form where the user can enter a name and search the LDAP server for that name.

Here's the form,

```
<html>
<head>
<title>Search</title>
</head>
```

```
<body>
<form action="search.php" method="POST">
<input type="text" name="name" length="30">
<input type="submit" name="submit" value = "Search">
</form>
</body>
</html>
```

and here's what it looks like:

And here's the code for the search script:

```
<html>
<head>
</head>

<body>

<?php

// specify the LDAP server to connect to
$conn = ldap_connect("localhost") or die("Could not connect to
server");

// bind to the LDAP server specified above
$r = ldap_bind($conn) or die("Could not bind to server");

// create the search string
$query = "(cn=" . $_POST['name'] . ")";

// start searching
// specify both the start location and the search criteria
// in this case, start at the top and return all entries $result =
ldap_search($conn,"dc=my-domain,dc=com", $query) or die ("Error in
search
query");

// get entry data as array
$info = ldap_get_entries($conn, $result);

// iterate over array and print data for each entry
echo "<ul>";
for ($i=0; $i<$info["count"]; $i++)
{
    echo "<li>" . $info[$i]["cn"][0] ." - ".$info[$i]["mail"][0] .
"</li>";
} echo "</ul>";

// print number of entries found
```

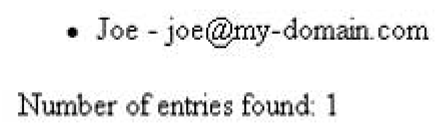
```
echo "Number of entries found: " . ldap_count_entries($conn, $result) .
"<p>";

// all done? clean up
ldap_close($conn);

?>

</body>
</html>
```

The only difference between this script and the previous one is that this time, the search criteria is dynamically generated using the data POSTed from the form.

```
$query = "(cn=" . $_POST['name'] . ")";
```

This parameter is then passed to the ldap_search() function and processed in the standard manner.

Here's what you should see, assuming that you searched for the string "joe":



- Joe - joe@my-domain.com

Number of entries found: 1

Simple when you know how, isn't it?

And that's about it for this first part. In the second part of this article, I will be showing you how to carry out more complex searches, and also add and delete information from the LDAP directory. Stay tuned for that one.and, until next time, stay healthy!

Note: Examples are illustrative only, and are not meant for a production environment. Melonfire provides no warranties or support for the source code described in this article. YMMV!