Using PHP with LDAP part two

By Harish Kamath

Table of Contents

Looking Back

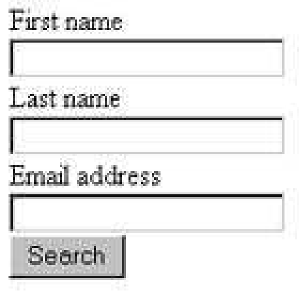
In the first part of this article, I gave you a crash course in the basics of using PHP with LDAP, demonstrating how PHP's built-in support for LDAP makes it easy to query an LDAP directory server for user information. That article also included a detailed discussion of the steps that make up a PHP-to-LDAP session, together with examples demonstrating how those steps play out in the real world. In this second (and concluding) article, I will carry the torch onwards, explaining how to perform more complex searches with LDAP. Since PHP comes with a whole bunch of functions for updating LDAP directory servers, I'll also examine those, with examples of how they can be used to add, modify and delete entries to the LDAP directory tree. Let's get going!

Of Needles And Haystacks

In the first part of this article, I had written a simple PHP script to search the LDAP directory for a particular user. But real life is never that simple...
Let’s suppose I want to search for my friend Joe from high school. Now, there are a million Joes in the world, and I would like to drill down to my Joe without having to navigate through a large set of results. The solution? A more complex search, which uses additional parameters to restrict the result set.
Take a look at the new search form,

```
<html><head><title>Search</title></head><body><form action="search.php" method="POST">First name<b
```

which looks like this:



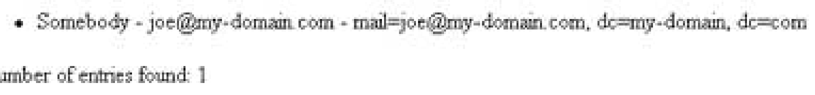
Once the form above is submitted, the data entered by the user is sent to the search script "search.php", which actually performs the query - take a look:

```
<html><head></head><body><?php// specify the LDAP server to connect to$conn = ldap_connect("localh
```

The structure of the code is identical to that of the examples in previous pages - with one important difference:

```
<?php// create the search string$query = "(&(cn=" . $_POST[’cn’] . ")(sn=" . $_POST[’sn’] . ")(mai
```

This search string, obviously with the variables replaced with actual values, is passed to the ldap_search() function; it returns only those entries from the LDAP directory free which match *all* the parameters specified. Why? Because of my usage of the special AND operator, signified by the addition of the ampersand (&) to the beginning of the search string above.
Here’s what the output looks like:



```
• Somebody - joe@my-domain.com - mail=joe@my-domain.com, dc=my-domain, dc=com

Number of entries found: 1
```

If your LDAP entries contain other attributes, it’s just as easy to create more complex search queries - simply add more input fields to the search form, and update the search string above to use those attributes when searching the directory tree.
In case you were wondering, yes, you can also use logical OR (the | operator) or logical NOT (the ! operator) in your search queries - I’ll leave that to you to play with.

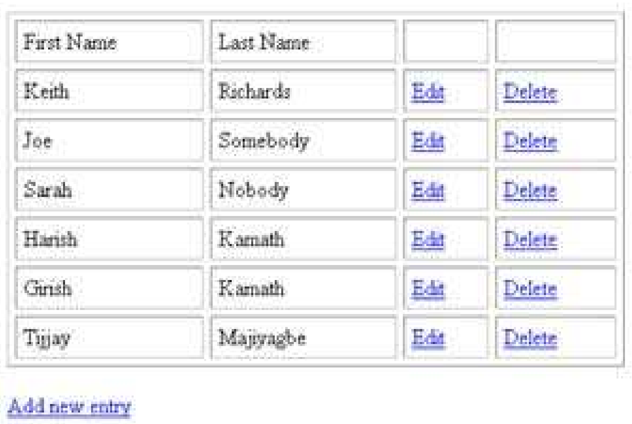
Making Lists

So that takes care of searching. Now, how about adding, editing and deleting entries?
PHP comes with a full-fledged API that allows easy modification of the LDAP directory tree. In order to demonstrate how this API works, I'm going to build, over the next few pages, a simple administration module that performs these functions, so that you can see how it's done.
First up, we need an index page that lists all the entries in the directory. This index page will serve as the starting point for an administrator to make changes to existing directory entries or add new ones. Here's the code,

---

```
  <html><head></head><body><table width="450" cellpadding="5" cellspacing="5" border="1"><?php// spe
```

and here's what it looks like:



First Name	Last Name		
Keith	Richards	Edit	Delete
Joe	Somebody	Edit	Delete
Sarah	Nobody	Edit	Delete
Harish	Kamath	Edit	Delete
Girish	Kamath	Edit	Delete
Tijjay	Majiyagbe	Edit	Delete

Add new entry

As you can see, most of this code is similar to what you saw in the previous article. However, there is one important difference - instead of using the ldap_search() function, I'm using the ldap_list() function, which returns a one-level list of all the entries matching the specified criteria, given a base DN at which to start searching.

```
  <?php// set base DN and required attribute list$base_dn = "dc=my-domain, dc=com";$params = array("
```

This base DN and search filter are provided to ldap_list() as second and third arguments respectively. In the example above, the ldap_list() function returns all the entries which have a "cn" attribute and are located immediately under the node with DN "dc=my-domain,dc=com".
Additionally, ldap_list() accepts a fourth, optional parameter - an array containing a list of all the attributes that should be included in the result set. In the example above, this array is called $params, and it specifies that the returned result set should contain the "cn", "sn" and "mail" attributes.
The search result identifier returned by the ldap_list() can be passed to the ldap_get_entries() function, which does the dirty work of extracting the raw data into a structured array. This array can be processed using a simple "for" loop.

```
  <?php// get entries$info = ldap_get_entries($conn, $result);// and print attribute valuesfor ($i=0
```

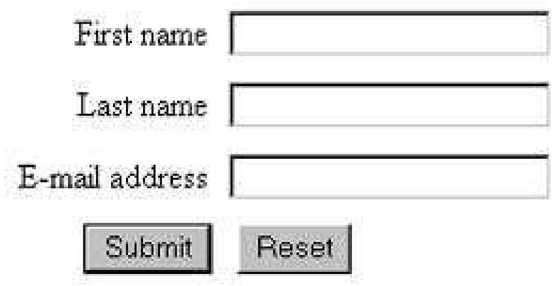
Note also the links to "edit.php" and "delete.php" next to each entry - I’ll be discussing the scripts these links point to shortly. For the moment, though, skip downwards to the last link on the page, which points to "add.html" - this is the HTML form that is used to add new users to the database, and it’s discussed on the next page.

Adding It All Up

Now that you know how to pull data out from the LDAP directory, how about putting some in?
PHP has a function to do this as well - I'll show you how to use it, by creating a simple interface to add instances of the "inetOrgPerson" class to the LDAP directory.
First, the input form, "add.html":

```
  <html><head><title>Add Entry</title></head><body><form method="POST" action="add.php">  <table bor
```

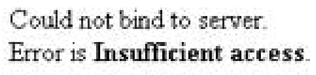
You'll notice here that I've only used three attributes of the "inetOrgPerson" class - "cn" for the common name, "sn" for the surname and "mail" for the email address. Feel free to add to this list if you like.
Here's what the form looks like,



and here's the script that actually adds the entry:

```
  <html><head></head><body><?php// specify the LDAP server to connect to$conn = ldap_connect("localh
```

Before I get into the details, let's give this code a quick test run. Enter some data into the form above and submit it - you will probably see something like this:


Could not bind to server.
Error is **Insufficient access**.

Ugly, huh?
In order to add an entry to the LDAP server, you must provide the server with appropriate credentials - something I've obviously not done in the example above. Typically, these credentials consist of the superuser's DN and password - information that you should have set when setting up your LDAP server.
Assuming you have this information, let's modify the code above and give it to the LDAP server.

```
  <html><head></head><body><?php// configure privileged user$root_dn = "cn=root, dc=my-domain, dc=co
```

Note the addition of user credentials in the call to ldap_bind() - these credentials will be used to authenticate the PHP client and allow it to make changes to the LDAP directory.

```
<?php // configure privileged user$root_dn = "cn=root, dc=my-domain, dc=com";$root_pw = "secret"; 
```

Note also that LDAP requires you to provide the complete DN of the superuser, not just the username (as is common with other authentication mechanisms).
Once that’s taken care of, the next step is to create an associative array whose keys correspond to attributes of an LDAP entry. The data for these attributes is obtained from the HTML form submitted by the user.

```
<?php// prepare data$info["cn"] = $_POST[’cn’];$info["sn"] = $_POST[’sn’];$info["mail"] = $_POST[’
```

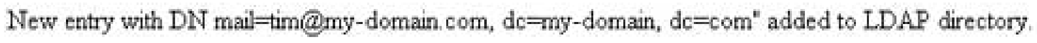
Once that’s done, I also need to construct the DN for the new entry. In this case, I’ve used the email address as a component of the entry’s DN in order to ensure uniqueness (LDAP DNs at the same level in the hierarchy must be unique).

```
<?php// prepare DN for new entry$dn = "mail=" . $_POST[’mail’] . ", dc=my-domain, dc=com";?>
```

In case you’re wondering where all this is going, you should know that all this information is needed by the ldap_add() functions, which is the PHP function that actually takes care of adding a new entry to the LDAP directory. This functions requires three arguments: a link identifier for the LDAP connection, the DN for the new entry, and the actual attributes of the entry. Since I now have all this in place, all that remains is to call ldap_add() and save the data to the LDAP server.

```
<?php// add data to directory$result = ldap_add($conn, $dn, $info);?>
```

And here’s what the result looks like:



New entry with DN mail=tim@my-domain.com, dc=my-domain, dc=com" added to LDAP directory.

In case you’re wondering about the numerous calls to ldap_error() in the code above, ignore them for the moment - I’ll be explaining them in detail shortly.

Changing Things Around

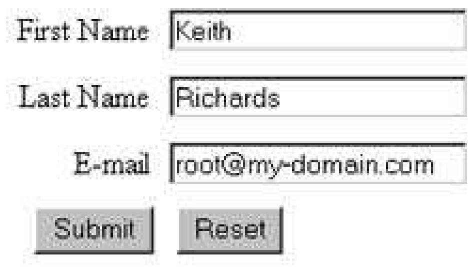
Next up, modifying entries. You might remember, from previous pages, that the index page of this application included links to scripts named "edit.php" and "delete.php" next to each entry, and passed each of those scripts certain data using the URL GET method. Here's what that code looked like:

```
  <td><a href=\"edit.php?mail=" . urlencode($info[$i]["mail"][0]) . "\">Edit</a></td>
```

As you can see, the user's email address is passed from the main index page to the "edit.php" script on the URL when the user clicks the corresponding hyperlink. This email address can then be used, in combination with the ldap_list() function, to retrieve the complete user record and display it in a form for editing - which is exactly what the next script does:

```
  <html><head></head><body><table width="450" cellpadding="5" cellspacing="5" border="1"> <?php// sp
```

Here's what it looks like:

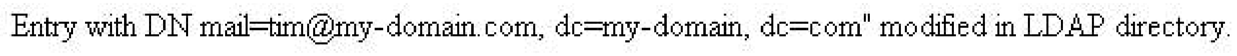


First Name	Keith
Last Name	Richards
E-mail	root@my-domain.com

I can now edit the data in the HTML form and submit it to a script that updates the entry in the directory. Here's the PHP code that take care of that:

```
  <html><head></head><body><?php// specify the LDAP server to connect to$conn = ldap_connect("localh
```

This code listing is similar to the code for adding a new user...with one important change: it uses the ldap_modify() function instead of the ldap_add() command to update an existing entry, rather than add a new one.
Here's the output of the script above, indicating that the update was successful:



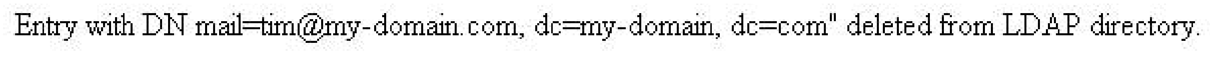
```
Entry with DN mail=tim@my-domain.com, dc=my-domain, dc=com" modified in LDAP directory.
```

Wiping Out The Past

So that takes care of listing, adding and modifying entries. All that’s left is to delete entries.
Again, this is similar to modifying entries - "delete.php", the script to invoke entry deletion is accessed from the main index page, and is passed the user’s email address using the URL GET method. This email address is then used by the PHP script to identify the corresponding entry and remove it via the ldap_delete() command.

```
  <html><head></head><body><?php// specify the LDAP server to connect to$conn = ldap_connect("localh
```

Here’s the output:



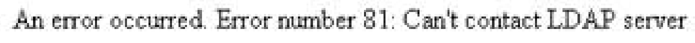
```
Entry with DN mail=tim@my-domain.com, dc=my-domain, dc=com" deleted from LDAP directory.
```

To Err Is Human...

You might remember, from the scripts in this article, my copious use of the ldap_error() and ldap_errno() functions. As you must have guessed by now, these are built-in API functions to record and display error messages.
The ldap_errno() function returns a pre-defined error number for each LDAP error. While this number is, by itself, not very useful, it acquires significance when coupled with yet another PHP function, ldap_err2str(), which returns a user-friendly error message for display to the user.
In order to see how this function may be used, consider the next example, which uses the ldap_error() and ldap_err2str() functions to trap and generate the error message resulting from an attempt to bind to a non-existent LDAP server:

```
<html><head></head><body><?php// specify the LDAP server to connect to$conn = ldap_connect("www.so
```

Here’s what the output looks like:


An error occurred. Error number 81: Can't contact LDAP server

There’s also a shortcut - the ldap_error() function, which returns the last error message generated. The following code snippet, which is equivalent to the one above, demonstrates:

```
<html><head></head><body><?php// specify the LDAP server to connect to$conn = ldap_connect("www.so
```

Endgame

And that’s about it for the moment. In this concluding article, I delved further into the PHP API for LDAP, demonstrating the PHP functions that let you retrieve and modify data in the LDAP directory tree. After a brief detour to demonstrate how you can easily build search forms for complex LDAP queries, I took you through the process of building a Web-based administration module for an LDAP directory, demonstrating the PHP API calls that allow you to add new entries to, and modify and delete existing entries from, the LDAP server. I also took a brief look at PHP’s error-handling capabilities in the LDAP world, demonstrating the two PHP functions that can be used to trap and gracefully handle errors in your PHP-LDAP scripts.

Obviously, this is just the beginning - LDAP has a whole bunch of different applications, and PHP is well-suited to all of them. The synergy between the two technologies cannot be overemphasized, and, as the LDAP API in PHP evolves, you can expect to see more and more applications building on this synergy.

In case you’d like to learn more about PHP and LDAP, I’d recommend some quality time with the following links:

Understanding LDAP, at http://www.devshed.com/Server_Side/Administration/LDAP

The PHP manual pages for LDAP, at http://www.php.net/manual/en/ref.ldap.php

The official OpenLDAP Web site, at http://www.openldap.org/

A discussion of OpenLDAP installation, at
http://www.newarchitectmag.com/documents/s=5641/new1013637552/sidebar2.htm

String representation of LDAP search filters, at http://www.ietf.org/rfc/rfc2254.txt

Until next time...be good!

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!