

Use PHP to create/modify Active Directory/LDAP entries

Use PHP to create and modify Active Directory or LDAP entries

By Shawn Grover
<http://grover.open2space.com>

December 31, 2008

Table of Contents

Introduction.....	3
Configuring the Active Directory Server.....	3
Create a storage container.....	4
Identify the distinguished name.....	4
Create the LDAP user account.....	4
Assign Permissions.....	5
Identify the contact attributes we want to use.....	5
*** A word about Schemas.....	6
Create a sample contact.....	6
The PHP side of things.....	7
Connecting to the server.....	7
Bind a username/password to the connection.....	7
Execute the LDAP operation.....	8
Searching.....	8
Adding an LDAP Entry.....	9
Modifying an LDAP Entry.....	10
Deleting an LDAP entry.....	12
Handling LDAP Errors.....	12
Working with the search results.....	13
Closing the connection.....	14
Wrapping up.....	15
Resources.....	19

Introduction

Working with [LDAP](#) servers from within a PHP based application can be a very handy trick to have up your sleeve. But the path to acquiring the know-how for this can be filled with mis-steps and confusing issues. Having recently worked through this, and finding the online documentation unhelpful for a relative LDAP novice, I'd like to document what I have found and help rectify that situation. The PHP documentation of the LDAP functionality is excellent, but knowing WHICH function to apply WHEN is not readily apparent - especially if you are doing something beyond a simple search. So in this article, we'll attempt to help fill in this information.

This article is not a definitive reference to LDAP. Rather we focus only on how to set up a working environment, and use PHP to access our server. This should be considered a launching point to digging deeper.

It should be noted that any PHP code for working against an [Active Directory](#) server should work with most any other LDAP servers - including [OpenLDAP](#). Of course some minor tweaks will be needed, but most of those are in the connection setup. If you are not interested in how to set up your Active Directory server for access, feel free to skip to "[The PHP side of things](#)" section.

We will work through the process of building parts of YACMS (Yet Another Contact Management System). We will assume an Active Directory (AD) server will provide the LDAP capabilities (with AD being an LDAP server itself with some customizations). We will also assume there is no need for our contacts to have network user accounts - this is just an address book, not a user management tool. We will begin with the necessary setup of the server explaining the background info as we go along, and then dive into the PHP code needed to add or modify contacts. We'll end things by putting together a PHP class to wrap up the interface for working with our contacts.

Let's get started.

Configuring the Active Directory Server

If you are following along closely, you'll need an operational Active Directory server and administrative access to it. Here's the basic steps for configuring the server:

- Create a storage container for our contacts
- Identify the "distinguished name" for that container
- Create a user account to manipulate our contact data
- Make sure that account has permission to add or modify the contacts
- Identify the contact attributes we want our application to use
- Create a sample contact

Warning: Most of the steps we indicate here are relatively harmless, but there may be a possibility that your Active Directory configuration could get messed up while you are working through this. A messed up directory can cause problems for your network - maybe even preventing logins. Please take care when working with your directory. You've been warned.

Create a storage container

We need to decide where we want to put our contacts. You can easily visualize this by opening "Active Directory Users and Computers" (found under Control Panel -> Administrative Tools). By default there is no clear place to put contacts. Where we store the contacts is arbitrary - you could store them under "Users", though this isn't recommended. Instead we will create a new "folder" here for our contacts. We'll call this folder "Address Book". We do this by right-clicking on your server name, selecting New, then selecting Organizational Unit. On the resulting dialog, type in "Address Book", and click OK. You will see an Address Book folder appear below your server name.

We are now beginning to run into the LDAP terminology. An Organization Unit (or OU for short) is exactly that - it allows you to organize your directory with something akin to a file folder. An OU can contain other OUs. If you want to separate companies from people, you might create a new OU below Address Book for "companies". Will stick with just the Address Book though.

Identify the distinguished name

The next important part we need to do is determine the "distinguished name" (or DN) for the Address Book OU. This is relatively simple. Let's assume your Active Directory domain is "example.local" (or it could be mycompany.com, or whatever you wanted). The domain name makes up part of the DN. A DN reads left to right, with the most specific item on the left, and the least specific item on the right. In our case now, our DN would be

```
OU=Address Book,DC=example,DC=local
```

The OU part just says which OU we want. If we had nested OUs (such as the "companies" mentioned above), then we just include that sub OU as well. For example "OU=companies,OU=Address Book,DC=example,DC=local". The DC parts (DC means Domain Component), simply identify the parts of the domain name.

This DN for the Address Book will become what is known as the "base DN" for our contacts.

Create the LDAP user account

This part is simple - just create a regular user account. I called mine "ldapuser". Remember the password you assign, we'll need that later when we connect to the server with PHP.

Now why we need the user account - that may not be so clear. Active Directory allows anonymous "read" access to the directory, but some items require specific privileges to view, and no changes can be made to the tree unless authenticated access is being used. Seeing as we want to read AND write, we need a user account.

Assign Permissions

To assign permissions to our OU, we'll have to turn on Advanced Features. In the "Active Directory Users and Computers" window, open the View menu, and click on (enable) Advanced Features.

Now, we can right click on our Address Book OU, and select Properties. Then click on the Security tab. There, we want to add our "ldapuser" account and make sure they have Read, Write, Create All Child Objects, and Delete All Child Objects access.

Note, you may need to go into Advanced options and adjust the permissions for the "ldapuser" to meet your needs. You will want to make sure those permissions propagate to all child objects.

Identify the contact attributes we want to use

An LDAP contact has many different fields that can be used - everything from the usual first/last name, address, phone number, etc., to the more specific "shipping address", or "secretary" field. In LDAP terminology, these fields are known as "attributes".

To see all the possible options, we have a few steps to go through:

1. Register the schema management tool - open a command prompt and type in "regsvr32 schmmgmt.dll". If you have any problems here, you may need to track down this DLL and/or specify the full path to it. You can find it under the i386 directory on the server install CD. This makes the schema management MMC snap-in available.
2. Open a new MMC window - go to Start -> Run, and type in "mmc" then click OK.
3. Add the snap-in - Go to the File Menu, choose "Add/Remove snap-in". Click the "Add" button on the resulting dialog. On the next dialog that opens, choose "Active Directory Schema". Click Add, then Close. Now click OK.
4. View the attributes - Expand the "Active Directory Schema" section that is now in the MMC window. Below that, expand the "Classes" item. Now scroll down on the left hand side until you find "contact" and select that.

You can see all the attributes available to you on the right hand side.

WARNING - DO NOT change anything using this tool. This would be the most dangerous point of this article. Getting the Schema's messed up could bork your Active Directory server quickly.

You may wish to save this MMC so that you can easily access the schemas at a later time.

It should be noted that you can assign values to any of the attributes listed (when setting values via PHP). However, some of those attributes are used or set by Active Directory itself - these attributes should be avoided. Some of the remaining attributes may not be visible under the Contact Properties dialog window - but we can still use these attributes if we'd like.

*** A word about Schemas

A schema is just the definition of what a class of objects will look like. There are many classes of objects - contact, inetOrgPerson, country, etc. These definitions are collectively known as "schemas" and may be combined to create more complex classes. A schema defines what attributes an object is allowed to have. For instance a contact may have a "secretary" property. Some of these attributes are required, while most are optional. The MMC snap-in described above shows the mandatory items, and will also show which schema defines the attribute.

Active Directory makes use of schemas to define how Active Directory runs. Heres the reason for the warning - do not change anything here. (yes, this is the "cover my ass" disclaimer)

Create a sample contact

We should create at least one contact record so we can check what PHP tells us about the LDAP record and infer how we should set up our data. While this is not really needed, it can be useful to improve interaction between the windows management tools and our web application. Right-click on Address Book (back in the Active Directory Users and Computers tool). From there choose New -> Contact. Fill in the first and last name fields in the resulting dialog and click OK.

Now, this creates our contact record, but doesn't really give it a lot of detail. Lets fill in ALL the details so that we can track down the attributes being used by this interface later. Right-click on the new contact, and select "Properties". Assign a unique value to all the fields on the General, Address, Telephones, and Organizations tabs. We can use these values later to find out what field name uses what attribute. (for instance, "First Name" is the "givenName" attribute)

When we get to the PHP code, we can do a search for all contacts and this sample record should be found. At that point we can dump the resulting data, via a **print_r()** command, to examine the structure and improve our understanding of how PHP interacts with the LDAP server.

The PHP side of things

We now should have a server that is set up up for us to work with it. Time to talk about the PHP code.

All LDAP operations in PHP follow a similar pattern:

- Connect to the server
- Bind a username/password to the connection (if needed)
- Execute the LDAP operation (search, add, modify, etc.)
- Close the server connection

Connecting to the server

First, you need to know the name or IP address of the target server. Keeping in mind that the system running the PHP code must be able to see that server - by name or by IP (whichever you choose to specify in this step). The PHP command is [ldap_connect\(\)](#):

```
$myConnection = ldap_connect("my_server");
```

This returns a resource variable that represents this connection - if the connection is made. This connection variable will be used in all other LDAP commands. If the connection fails, false is returned.

If you are connecting to an Active Directory server, you'll likely need to adjust some of the options before much else will work properly. This is done using the [ldap_set_option\(\)](#) function. In my case, I needed the following:

```
ldap_set_option($myConnection, LDAP_OPT_PROTOCOL_VERSION, 3);
```

Bind a username/password to the connection

Now that we have a connection to the server, we need to tell the server which user we want to perform our operations as. This is known as "binding". Now the secret part here is that most LDAP servers allow anonymous read access by default. So, if you are just looking up info, you may not need to do a binding at all. But the moment you want to modify things, or to access anything that has been explicitly restricted, you'll need to bind the username/password to use. The command to do the binding is [ldap_bind\(\)](#):

```
ldap_bind( $ldapConn, $myUsername, $myPassword );
```

`ldap_bind` will return true if successful, and false if not. Using this info, we can do an easy LDAP authentication by trying to bind with the user supplied username/password. If you get a true response, then the username and password combination are valid.

Execute the LDAP operation

Now the "meat" of the code. We'll start simple with the search command.

Searching

The PHP command to do an LDAP search is [ldap_search\(\)](#). This function takes a few arguments that we need to explain further.

- Link Identifier - this is just a fancy way to say our connection variable. If we created a connection with the code above, we'd use \$myConnection here.
- Base DN - Remember at the start of the article, we created a "container", and determined it's Distinguished Name? That would be the "Base DN" if we were searching for items within that container (such as contacts!). The Base DN basically says to "start at this point in the LDAP tree, and search the child branches only". The actual value you place here depends on your needs, but the more specific you can make the Base DN, the faster your searches will be.
- Filter - The filter defines what it is we are looking for. If we just wanted all the contacts within the specified base DN, then we could say "(cn=*)" here. You can indicate any attribute you want to search by, and the expected value - with wildcards even. So if we were looking for people whose last name started with G, then our filter might be "(sn=G*)". You can also combine filter conditions with AND or OR operations. You can learn more about LDAP filters at <http://confluence.atlassian.com/display/DEV/How+to+write+a+LDAP+search+filter>.
- Attributes - This is optional. If you do not specify an array of desired attributes, you will get back ALL attributes for ALL elements that match the filter below the Base DN. So if you were just looking for the phone numbers for "Bob Smith", you might set the filter to "(cn=Bob Smith)", and then specify an attributes array of array("telephonenumber"). Returning fewer attributes results in faster searches.

There are two other optional parameters - sizelimit and timelimit. I have not had need to use these (yet) so cannot really comment on them. They do what they sound like - limit how many results you'll get back, or how long the search will be allowed to run for (in seconds). If either are set to zero, there is no limit for that item. Or more to the point, the default limits on the LDAP server are used.

The ldap_search will return false if it fails. A search may look like this:

```
$base_dn = 'OU=Address Book,DC=example,DC=local';
$filter = '(cn=*)';
$attr = array('cn', 'givenname', 'sn', 'description', 'distinguishedname');
$result = ldap_search($myConnection, $base_dn, $filter, $attr);
if ( $result ) {
    $entries = ldap_get_entries( $myConnection, $result );
    return $entries;
}
```

Use PHP to create/modify Active Directory/LDAP entries

Here we are setting up handy variables to hold our parameter information, executing the search, and then checking if we have any results. If we DO have results, we are making use of the [ldap_get_entries\(\)](#) function. This function converts the results into a multi-dimensional array that we can then use in code. A "print_r(\$entries);" before the return statement is handy when starting out with LDAP.

Adding an LDAP Entry

Adding an LDAP entry is conceptually easy, but a little more difficult in practice. What makes it difficult is that you need to get the new data into just the right format. This is sometimes challenging. I have found the easiest method is to query for a similar record, do a print_r() of that entry, then try to recreate a similar array structure for the new entry. I'll take you through a sample just to be clear though because there are still some gotchas.

The command for adding an ldap entry is [ldap_add\(\)](#). It takes 3 parameters:

- Link Identifier - our connection variable - \$myConnection, for example.
- DN - the distinguished name for the new entry. This can be a little confusing at first. First the easy part. It's kinda arbitrary. The DN needs to be unique, but beyond that we could, in theory, use any attributes we want. Now the not so easy part. In practice, this depends on what "object class" our new entry is, and Active Directory seems to demand a specific format for contacts - CN=,OU=,DC=,DC=. Specifically, Active Directory appears to require the CN attribute be used in the DN. See the sample below for a little more detail.
- Entry - an array with key => value pairs - one for each attribute of our entry.

Sounds simple enough, but wait.. there's more! Part of the attributes MUST be the "objectclass(s)" for the entry. (the class is just a 'type' of entry.) For instance, a typical contact has object types of "top", "person", "organizationalPerson", and "contact". Each of these object classes provides various attributes we may want to use. If we wanted other attributes declared in other classes, we would specify those classes here. We use a multi-dimensional array with this info, as part of our entry attributes:

```
$entry["objectClass"][0] = "top";  
$entry["objectClass"][1] = "person";  
$entry["objectClass"][2] = "organizationalPerson";  
$entry["objectClass"][3] = "contact";
```

Now, if you are doing print_r() of an existing record, keep in mind that there are a bunch of attributes that Active Directory (or Open LDAP for that matter) may add on it's own. It is unlikely that you will need to add an "SSID", or similar attribute manually. Use the schema viewer (described above) to get a hint of the mandatory and optional values. You may also be able to get a hint as to which ones Active Directory will populate for you. But, if you are looking at typical attributes for a contact (name, address, phone, email, etc.) you should be fine. With the exception of the objectTypes.

So we know the DN for the entry, and the Object Types. But we must also add the CN attribute for a contact, in addition to the first and last names. So we would need to do some string concatenation to build the CN. It is recommended that the CN value match that used in the DN. (I have found they CAN be different, but that just leads to confusion and errors.)

Use PHP to create/modify Active Directory/LDAP entries

```
$entry["cn"] = $firstname . " " . $lastname;  
$entry["givenname"] = $firstname;  
$entry["sn"] = $lastname;
```

Now, once we have carefully formulated our \$entry array, we can add it to the LDAP tree:

```
$result = ldap_add($myConnection, $dn, $entry);
```

ldap_add will return false if it fails. See the section below on handling LDAP errors for how a false condition may be taken care off.

You may find that the ldap_add() function results in a few warnings, even though your array is correct. Some of these warnings may result from putting an empty string into your entry array. In this case, you can tell PHP to NOT display the warnings by putting an @ sign in front of the command:

```
$result = @ldap_add($myConnection, $dn, $entry);
```

Modifying an LDAP Entry

There are a LOT of resources on the web that tell you how to search and insert records. But I found very few that tell you how to modify an existing record. The process is similar to adding a record but has some noticeable differences.

The first thing we need is the Distinguished Name (DN) of the record we want to update. This is akin to knowing the ID of which database record to update.

Next we need to format our data array properly. The data is the same as for the insert routine, however any empty array values need to be removed. This code does the trick nicely:

```
foreach ($data as $key => $value) {  
    if (empty($value)) {  
        unset($data[$key]);  
    }  
}
```

The confusing part with updating an LDAP record is the way the functions are named. There is [ldap_mod_replace\(\)](#), [ldap_rename\(\)](#), and [ldap_modify\(\)](#) that all look like possible candidates for our update. In practice, you'll need a combination of these to do the update. Especially if you do something silly like let the values that make up the Common Name (CN) get modified. You know, first name, last name, etc. (Ok, sarcasm is turned off now...) Active Directory does not like this at all.

The problem is what is known as the Relative Distinguished Name, or RDN. This is a combination of the base DN, and the common name. Active Directory makes use of the RDN while updating records. So if we try to modify the elements that make up the RDN, Active Directory has fits.

Use PHP to create/modify Active Directory/LDAP entries

What I found to work is the following steps:

1. Regenerate the CN string
2. Rename the record with the new CN (via `ldap_rename`)
3. Determine the revised DN for the record
4. Replace the attribute values (via `ldap_mod_replace`)

Code for this might look like the following:

```
// $data is our array of attributes/values
$cn = "cn=". $data["givenname"] ." ". $data["sn"];
$dn = $data["dn"];
$changed = ldap_rename($myConnection, $dn, $cn, null, true);
if ($changed) {
    $base_dn = 'OU=Address Book,DC=example,DC=local';
    $newdn = $cn .",". $base_dn;
    ldap_mod_replace($myConnection, $newdn, $data);
}
```

Let's explore the LDAP functions a little more.

- [ldap_rename](#) - We call this function because it is reasonable for the first name and last name to be changed, which in turn changes our RDN. If we tried to simply modify a record with a changed name, then we would see errors. This function is passed 5 arguments.
 - `$myConnection` - the connection to the ldap server
 - `$dn` - the distinguished name for the record, before changes
 - `$cn` - the NEW or revised Common Name for the record. HINT: There is no rule that says the CN **must** be different...
 - `null` - While we have a null, you can specify a new parent for the record. You would place a value here if you were moving the record into another Organizational Unit. (think about moving a file into a different directory).
 - `true` - this is a True/False value indicating if the old record (the one before any changes) should be removed. I imagine you'd want to set this to false if you were copying a contact into multiple OU's, but I have not needed to do this yet.
- [ldap_mod_replace](#) - Because we are allowing the potential change of first name and last name, we need to update the other items independently. If we were not allowing a potential change to the values that make up the RDN, we could likely just use `ldap_modify`. (i.e. we were changing a phone number...) This function is passed three parameters.
 - `$myConnection` - the connection to the ldap server
 - `$newdn` - the re-calculated DN for the record in question. This would catch a revised CN if the first/last names were changed.
 - `$data` - our data array containing key/value pairs for the attributes that need to be changed. Because the firstname and sn values now properly match the RDN, we can safely leave those values in the array.

Use PHP to create/modify Active Directory/LDAP entries

The only real improvement to be made here is to handle if the `$changed` variable is NOT true - the rename did not work. An error could be thrown, or an alert box can be displayed. The particulars of WHY it failed is another matter though. See the Errors section below for more details.

Deleting an LDAP entry

Deleting an LDAP record is very straight forward - as long as you know the Distinguished Name. We make use of the [ldap_delete\(\)](#) function. This function takes two arguments - the connection object, and the distinguished name to remove. The function will return false if it fails, so we can raise an error in such a condition if needed.

```
$removed = ldap_delete($myConnection, $dn);
```

One note - there is NO warning or easy recovery from the delete operation. Be sure to prompt your users for confirmation before doing a delete, where appropriate.

Handling LDAP Errors

Things fail. It happens. Good code takes this into account and tries to handle these failures in a reasonable manner. LDAP is no different - things will fail at some point. Luckily handling the errors is fairly simple. The functions we've covered so far will return false when things go wrong. To find out what went wrong we make use of the [ldap_errno\(\)](#) function and the [ldap_error\(\)](#) function. Both of these functions take a single argument - the connection object.

- [ldap_errno\(\)](#) - This function returns the error number for the last error that occurred.
- [ldap_error\(\)](#) - this function returns a string message for the last error that occurred.

So with that in mind, we can raise an error, show an error message, or even just log the details if an LDAP operation fails.

```
$removed = ldap_delete($myConnection, $dn);
if (!$removed) {
    throw new Exception(
        "Error #". ldap_errno($myConnection) . " : ".ldap_error($myConnection)
    );
}
```

Working with the search results

An LDAP search returns an object containing the results. Sometimes this is not very convenient to work with. While the [ldap_get_entries\(\)](#) function will convert the results to an array, it is sometimes not as convenient as building our own result array. Or sometimes we need to process the results directly. We'll explore converting the results to a custom array, but the ideas introduced here can be applied any time you need to loop through the results quickly. At a very minimum, understanding these functions and how to use them will give us greater insight into the structure of the LDAP results.

To loop over our result entries, we require a few more functions than we've seen thus far. The idea is you start with the first result record, then the first attribute. Next you loop through the attributes until there are none left, putting this attribute and value into our result array. Then we move on to the next record, and repeat the process until there is no data left. This needs a number of functions:

- [ldap_first_entry\(\)](#) - this will get the first result entry. It takes two parameters - the LDAP connection object, and the result set in question.
- [ldap_first_attribute\(\)](#) - this will get the first attribute of the entry in question. It also takes two arguments - the connection object, and the entry or result set.
- [ldap_get_values_len\(\)](#) - find the value of a given attribute in a given entry. There are three parameters this time - the ldap connection object, the entry we are working with, and the attribute we want the value of. Alternatively you could use the [ldap_get_values\(\)](#) function which is the same thing, with the same parameters - the difference is that the [ldap_get_values_len](#) function handles binary data.
- [ldap_next_attribute\(\)](#) - move to the next attribute. The function takes two parameters - the connection object, and the entry being examined.
- [ldap_next_entry\(\)](#) - move to the next entry in the result set. Two parameters here - the connection object and the current entry.

It should be noticed that the connection object is used throughout, and it maintains some state information regarding where in the result set and the attributes list we are. Making use of these functions we can loop through each of the results. One note though - getting the value of the current attribute can be a little tricky. In some cases the attribute may contain an array of values - even if there is only one entry. We need to see if there are more than one items, and if so we'll just put this value array into our results, otherwise we'll get the value of the first (and only) entry and store that in our results.

Use PHP to create/modify Active Directory/LDAP entries

A sample routine may look like this:

```
$base_dn = 'OU=Address Book,DC=example,DC=local';
$filter = '(cn=*)';
$attr = array('cn', 'givenname', 'sn', 'description', 'distinguishedname');
$result = ldap_search($myConnection, $base_dn, $filter, $attr);
$resultArray = array();
if ($result) {
    $entry = ldap_first_entry($myConnection, $result);
    while ($entry) {
        $row = array();
        $attr = ldap_first_attribute($myConnection, $entry);
        while ($attr) {
            $val = ldap_get_values_len($myConnection, $entry, $attr);
            if (array_key_exists('count', $val) AND $val['count'] == 1) {
                $row[strtolower($attr)] = $val[0];
            }
            else {
                $row[strtolower($attr)] = $val;
            }
            $attr = ldap_next_attribute($myConnection, $entry);
        }
        $resultArray[] = $row;
        $entry = ldap_next_entry($myConnection, $entry);
    }
}

print_r($resultArray);
```

The last line here simply dumps our array to the screen for us to inspect. Of course we can do anything we want with that array, such as applying our presentation or business logic. Or we could apply that logic within the loops if needed.

Closing the connection

The final thing we need to do is close the connection to the LDAP server. This is very simple - we call the [ldap_unbind\(\)](#) function. It takes a single parameter - the ldap connection object. Once closed no further LDAP functions can be called on that connection object. If it's easier to think of, there is also the [ldap_close\(\)](#) function - this is an alias for [ldap_unbind\(\)](#) and takes the same arguments.

```
ldap_unbind($myConnection); // OR ldap_close($myConnection);
```

Wrapping up

Phew! That's a lot of material to go through. Let's summarize the code with a nice simple class that wraps up all the functionality we've just discussed. (NOTE: Some of the lines had to be tweaked to fit in a reasonable way. Watch for this if you are cutting/pasting the code.)

```
<?php
class Contacts {
    // we'll store the LDAP connection so we
    // do not need to re-connect for every command
    private $connection = null;

    /**
     * Connect and bind to the LDAP or Active Directory Server
     * NOTE: we are assuming the default port of 389.
     * Alternate ports should be specified in the ldap_connect function,
     * if needed. * NOTE: We are using the singleton pattern here - we only
     * create a connection if it does not exist.
     */
    public function connect($server, $user, $password) {
        if ($this->connection) {
            return $this->connection;
        }
        else {
            $ldapConn = ldap_connect($server);
            if ( $ldapConn ) {
                ldap_set_option($ldapConn, LDAP_OPT_PROTOCOL_VERSION, 3);
                if ( ldap_bind( $ldapConn, $user, $password ) ) {
                    $this->connection = $ldapConn;
                    return $this->connection;
                }
            }
        }
    }

    /**
     * Search an LDAP server
     */
    public function search($basedn, $filter, $attributes) {
        $connection = $this->connect();
        $results = ldap_search($connection, $basedn, $filter, $attributes);
        if ($results) {
            $entries = ldap_get_entries($connection, $results);
            return $entries;
        }
    }
}
```

Use PHP to create/modify Active Directory/LDAP entries

```
/**
 * Add a new contact
 */
public function add($basedn, $firstname, $lastname, $address, $phone) {
    //set up our entry array
    $contact = array();
    $contact['objecttype'][0] = 'top';
    $contact['objectclass'][1] = 'person';
    $contact['objectclass'][2] = 'organizationalPerson';
    $contact['objectclass'][3] = 'contact';

    //add our data
    $contact['givenname'] = $firstname;
    $contact['sn'] = $lastname;
    $contact['streetaddress'] = $address;
    $contact['telephonenumber'] = $phone;

    //Create the CN entry
    $contact['cn'] = $firstname . ' ' . $lastname;

    //create the DN for the entry
    $dn = 'cn=' . $contact['cn'] . ', ' . $basedn;

    //add the entry
    $connection = $this->connect();
    $result = ldap_add($connection, $dn, $contact);
    if (!$result) {
        //the add failed, lets raise an error and hopefully find out why
        $this->ldapError();
    }
}

/**
 * Modify an existing contact
 */
public function modify($basedn, $dnToEdit,
    $firstname, $lastname, $address, $phone) {
    //get a reference to the current entry
    $connection = $this->connect();
    $result = ldap_search($connection, $dnToEdit);
    if (!$result) {
        // the search failed
        $this->ldapError();
    }

    //convert the results to an array for easier use.
    $contact = $this->resultToArray($result);
}
```

Use PHP to create/modify Active Directory/LDAP entries

```
//set the new values
$contact['givenname'] = $firstname;
$contact['sn'] = $lastname;
$contact['streetaddress'] =
$address; $contact['telephonenumber'] = $phone;

//remove any empty entries
foreach ($contact as $key => $value) {
    if (empty($value)) {
        unset($contact[$key]);
    }
}

//Find the new CN - in case the first or last name has changed
$cn = 'cn='. $firstname .' '. $lastname;

//rename the record (handling if the first/last name have changed)
$changed = ldap_rename($connection, $dnToEdit, $cn, null, true);
if ($changed) {
    //find the DN for the potentially revised name
    $newdn = $cn .','.' $basedn;

    //now we can apply any changes in the contact information
    ldap_mod_replace($connection, $newdn, $contact);
}
else {
    $this->ldapError();
}
}

/**
 * Remove an existing contact
 */
public function delete($dnToDelete) {
    $connection = $this->connect();
    $removed = ldap_delete($connection, $dnToDelete);
    if (!$removed) {
        $this->ldapError();
    }
}

/**
 * throw an error, getting the needed info from the connection object
 */
public function ldapError() {
    $connection = $this->connect();
    throw new Exception(
        'Error: ('. ldap_errno($connection) .') '. ldap_error($connection)
    );
}
```

Use PHP to create/modify Active Directory/LDAP entries

```
/**
 * Convert an LDAP search result into an array
 */
public function resultToArray($result) {
    $connection = $this->connect();
    $resultArray = array();
    if ($result) {
        $entry = ldap_first_entry($connection, $result);
        while ($entry) {
            $row = array();
            $attr = ldap_first_attribute($connection, $entry);
            while ($attr) {
                $val = ldap_get_values_len($connection, $entry, $attr);
                if (array_key_exists('count', $val) AND $val['count'] == 1) {
                    $row[strtolower($attr)] = $val[0];
                }
                else {
                    $row[strtolower($attr)] = $val;
                }
                $attr = ldap_next_attribute($connection, $entry);
            }
            $resultArray[] = $row;
            $entry = ldap_next_entry($connection, $entry);
        }
    }
    return $resultArray;
}

/**
 * disconnect from the LDAP server
 */
public function disconnect() {
    $connection = $this->connect();
    ldap_unbind($connection);
}
}
?>
```

Use PHP to create/modify Active Directory/LDAP entries

And with that, we may make use of that class as follows:

```
$contacts = new Contacts();
$contacts->add(
    "OU=Address Book,DC=example,DC=local",
    "Mike",
    "Smith",
    "123 Main Street",
    "(555) 555-5555"
);
$contacts->disconnect();
```

The code is just a sample to see how things might be implemented. As I sit here, I can see many ways to improve the code, but I'll leave that to you. The idea here was to learn about working with LDAP, not how to optimize the PHP code. :)

I hope this has proven helpful, and perhaps we'll see more LDAP applications hit the interwebs as a result.

Resources

- [PDF version of this article](#)
- [The class definition](#) - same as above, but as a file. This is done as a text file - be sure to rename it.
- [PHP LDAP functions](#) – the source to find the definitions of the functions and their parameters.