Zend_Ldap - Michael B. Allen

<ac:macro ac:name="info"><ac:parameter ac:name="title">Minimal Functionality</ac:parameter><ac:rich-text-body>
Currently this class is designed only to satisfy the limited functionality necessary for the Zend_Auth_Adapter_Ldap authentication adapter. Operations such as searching, creating, modifying or renaming entries in the directory are currently not supported and will be defined at a later time. </ac:rich-text-body></ac:macro>

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Ldap Component Proposal

Proposed Component Name	Zend_Ldap
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Ldap
Proposers	Michael B Allen Darby Felton, Zend liaison
Revision	(wiki revision: 15)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
Automatic Username Canonicalization When Binding
Zend_Ldap Options
Account Name Canonicalization
Multi-domain Authentication and Failover
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Ldap is proposed as a class for performing LDAP operations including but not limited to binding, searching and modifying entries in an LDAP directory.

2. References

- The implementation on which this proposal is based is currently available here: http://www.ioplex.com/code/. This code implements everything described in this document.
- The PHP LDAP API.
- RFC 4510 - Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map
- RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol
- Zend_Auth_Adapter_Ldap

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **should** throw exceptions for configuration errors, environmental issues and invalid usage (e.g. required options missing, ldap extension unavailable, wrong parameter supplied to method, etc).
- Currently this component **will** use the standard ldap extension shipped with PHP.
- This component **must** support optional SSL / TLS encrypted transport.

4. Dependencies on Other Framework Components

- `Zend_Ldap_Exception` - `Zend_Ldap_Exception` is a companion class to `Zend_Ldap` used to handle unexpected LDAP extension errors and LDAP specific protocol errors (e.g. failed to connect to LDAP server).
- `Zend_Exception` - `Zend_Ldap_Exception` extends `Zend_Exception`

5. Theory of Operation

This component currently consists of two classes `Zend_Ldap` and `Zend_Ldap_Exception`. The `Zend_Ldap` class conceptually represents a binding to a single LDAP server. The parameters for binding may be provided explicitly or in the form of an options array.

Using the `Zend_Ldap` class depends on the type of LDAP server and is best summarized with some simple examples.

If you are using OpenLDAP a simple example looks like the following (note that the `bindRequiresDn` option is important if you are *not* using AD):

```
require_once 'Zend/Ldap.php';

$options = array(
    'host' => 's0.foo.net',
    'username' => 'CN=user1,DC=foo,DC=net',
    'password' => 'pass1',
    'bindRequiresDn' => true,
    'accountDomainName' => 'foo.net',
    'baseDn' => 'OU=Sales,DC=foo,DC=net',
);
$ldap = new Zend_Ldap($options);
$acctname = $ldap->getCanonicalAccountName('abaker', Zend_Ldap::ACCTNAME_FORM_DN);
echo "$acctname\n";
```

If you are using Microsoft AD a simple example is:

```
require_once 'Zend/Ldap.php';

$options = array(
    'host' => 'dc1.w.net',
    'useSsl' => true,
    'username' => 'user1@w.net',
    'password' => 'pass1',
    'accountDomainName' => 'w.net',
    'accountDomainNameShort' => 'W',
    'baseDn' => 'CN=Users,DC=w,DC=net',
);
$ldap = new Zend_Ldap($options);
$acctname = $ldap->getCanonicalAccountName('bcarter', Zend_Ldap::ACCTNAME_FORM_DN);
echo "$acctname\n";
```

Note that we use the `getCanonicalAccountName()` method to retrieve the account DN here only because that is what exercises the most of what little code is currently present in this class.

Automatic Username Canonicalization When Binding

If `bind()` is called with a non-DN username but `bindRequiresDN` is `true` and no username in DN form was supplied as an option, the bind will fail. However, if a username in DN form is supplied in the options array, `Zend_Ldap` will first bind with that username, retrieve the account DN for the username supplied to `bind()` and then re-bind with that DN.

This behavior is critical to Zend_Auth_Adapter_Ldap which passes the username supplied by the user directly to `bind()`.

The following example illustrates how the non-DN username 'abaker' can be used with `bind()`:

```
require_once 'Zend/Ldap.php';

$options = array(
        'host' => 's0.foo.net',
        'username' => 'CN=user1,DC=foo,DC=net',
        'password' => 'pass1',
        'bindRequiresDn' => true,
        'accountDomainName' => 'foo.net',
        'baseDn' => 'OU=Sales,DC=foo,DC=net',
);
$ldap = new Zend_Ldap($options);
$ldap->bind('abaker', 'moonbike55');
$acctname = $ldap->getCanonicalAccountName('abaker', Zend_Ldap::ACCTNAME_FORM_DN);
echo "$acctname\n";
```

The `bind()` call in this example sees that the username 'abaker' is not in DN form, finds `bindRequiresDn` is `true`, uses 'CN=user1,DC=foo,DC=net' and 'pass1' to bind, retrieves the DN for 'abaker', unbinds and then rebinds with the newly discovered 'CN=Alice Baker,OU=Sales,DC=foo,DC=net'.

Zend_Ldap Options

The `Zend_Ldap` component accepts an array of options either supplied to the constructor or through the `setOptions()` method. The permitted options are as follows:

Name	Description
host	The default hostname of LDAP server if not supplied to `connect()` (also may be used when trying to canonicalize usernames in `bind()`).
port	Default port of LDAP server if not supplied to `connect()`.

useSsl	Whether or not the LDAP client should use SSL / TLS encrypted transport. A value of true is strongly favored in production environments to prevent passwords from be transmitted in clear text. The default value is `false` as servers frequently require that a certificate be installed separately after installation.
username	The default credentials username. Some servers require that this be in DN form.
password	The default credentials password (used only with username above).
bindRequiresDn	If `true`, this instructs `Zend_Ldap` to retrieve the DN for the account used to bind if the username is not already in DN form. The default value is `false`.
baseDn	The default base DN used for searching (e.g. for accounts). This option is required for most account related operations and should indicate the DN under which accounts are located.
accountCanonicalForm	A small integer indicating the form account names should be canonicalized to. See the *Account Name Canonicalization* section below.
accountDomainName	The FQDN domain for which the target LDAP server is an authority (e.g. example.com).
accountDomainNameShort	The 'short' domain for which the target LDAP server is an authority. This is usually used to specify the NetBIOS domain name for Windows networks but may also be used by non-AD servers.
accountFilterFormat	The LDAP search filter used to search for accounts. This string is a `printf` style expression that must contain one `'%s'` to accomodate the username. The default value is `'(&(objectClass=user)(sAMAccountName=%s))'` unless `bindRequiresDn` is set to `true` in which case the default is `'(&(objectClass=posixAccount)(uid=%s))'`. Users of custom schemas may need to change this option.

Account Name Canonicalization

The `accountDomainName` and `accountDomainNameShort` options are used for two purposes - 1) they facilitate multi-domain authentication and failover capability and 2) they are also used to canonicalize usernames. Specifically, names are canonicalized to the form specified by the `accountCanonicalForm` option. This option may one of the following values:

Name	Value	Example
`ACCTNAME_FORM_DN`	1	CN=Alice Baker,CN=Users,DC=example,DC=com
`ACCTNAME_FORM_USERNAME`	2	abaker
`ACCTNAME_FORM_BACKSLASH`	3	EXAMPLE\abaker
`ACCTNAME_FORM_PRINCIPAL`	4	abaker@example.com

The default canonicalization depends on what account domain name options were supplied. If `accountDomainNameShort` was supplied the default `accountCanonicalForm` value is `ACCTNAME_FORM_BACKSLASH`. Otherwise, if `accountDomainName` was supplied, the default is `ACCTNAME_FORM_PRINCIPAL`.

Account name canonicalization ensures that the string used to identify an account is consistent regardless of what was supplied to `bind()`. For example, if the user supplies an account name of *abaker@example.com* or just *abaker* and the `accountCanonicalForm` is set to 3, the resulting canonicalized name would be *EXAMPLE\abaker*.

Multi-domain Authentication and Failover

The `Zend_Ldap` component by itself makes no attempt to authenticate with multiple servers. However, `Zend_Ldap` is specifically designed to handle this scenario gracefully. The required technique is to simply iterate over an array of arrays of server options and attempt to bind with each server. As described above `bind()` will automatically canonicalize each name so it does not matter if the user passes `abaker@foo.net` or `W\bcarter` or `cdavis` - the `bind()` method will only succeed if the credentials were successfully used in the bind.

Consider the following example that illustrates the technique required to implement multi-domain authentication and failover:

```
$acctname = 'W\\user2';
$password = 'pass2';

$multiOptions = array(
    'server1' => array(
        'host' => 's0.foo.net',
        'username' => 'CN=user1,DC=foo,DC=net',
        'password' => 'pass1',
        'bindRequiresDn' => true,
        'accountDomainName' => 'foo.net',
        'accountDomainNameShort' => 'FOO',
        'accountCanonicalForm' => 4, // ACCT_FORM_PRINCIPAL
        'baseDn' => 'OU=Sales,DC=foo,DC=net',
    ),
    'server2' => array(
        'host' => 'dc1.w.net',
        'useSsl' => true,
        'username' => 'user1@w.net',
        'password' => 'pass1',
        'accountDomainName' => 'w.net',
        'accountDomainNameShort' => 'W',
        'accountCanonicalForm' => 4, // ACCT_FORM_PRINCIPAL
        'baseDn' => 'CN=Users,DC=w,DC=net',
    ),
);

$ldap = new Zend_Ldap();

foreach ($multiOptions as $name => $options) {

    echo "Trying to bind using server options for '$name'\n";

    $ldap->setOptions($options);
    try {
        $ldap->bind($acctname, $password);
        $acctname = $ldap->getCanonicalAccountName($acctname);
        echo "SUCCESS: authenticated $acctname\n";
        return;
    } catch (Zend_Ldap_Exception $zle) {
        echo '  ' . $zle->getMessage() . "\n";
        if ($zle->getCode() === Zend_Ldap_Exception::LDAP_X_DOMAIN_MISMATCH) {
            continue;
        }
    }
}
```

If the bind fails for any reason, the next set of server options is tried.

The `getCanonicalAccountName` call gets the canonical account name that the application would presumably use to associate data with such as preferences. The `accountCanonicalForm = 4` in all server options ensures that the canonical form is consistent regardless of which server was ultimately used.

The special `LDAP_X_DOMAIN_MISMATCH` exception occurs when an account name with a domain component was supplied (e.g. `abaker@foo.net` or `FOO\abaker` and not just `abaker`) but the domain component did not match either domain in the currently selected server options. Meaning, this exception indicates that the server is not an authority for the account. In this case, the bind will not be performed therby eliminating unnecessary communication with the server. Note that the `continue` instruction has no effect in this example but in practice for error

handling and debugging purposes you will probably want to check for `LDAP_X_DOMAIN_MISMATCH` as well as `LDAP_NO_SUCH_OBJECT` and `LDAP_INVALID_CREDENTIALS`.

The above code is very similar to that of Zend_Auth_Adapter_Ldap. In fact, we recommend that you simply use that authentication adapter for multi-domain + failover LDAP based authentication (or copy the code).

6. Milestones / Tasks

- Milestone 1: [DONE] Create initial prototype.
- Milestone 2: [DONE] Create documentation necessary to use and test prototype.
- Milestone 3: Working prototype checked into the incubator.
- Milestone 4: Create unit tests

7. Class Index

- Zend_Ldap
- Zend_Ldap_Exception

8. Use Cases

See examples herein. Currently this class is used exclusively by the Zend_Auth_Adapter_Ldap authentication adapter and as such there no use-cases other than those associated with binding (used as an authentication mechanism) and account name canonicalization.

9. Class Skeletons

```
class Zend_Ldap
{
    const ACCTNAME_FORM_DN        = 1;
    const ACCTNAME_FORM_USERNAME  = 2;
    const ACCTNAME_FORM_BACKSLASH = 3;
    const ACCTNAME_FORM_PRINCIPAL = 4;

    public static function filterEscape($str);
    public function __construct($options);
    public function setOptions($options);
    public function getResource();
    public function getCanonicalAccountName($acctname, $form = 0);
    public function disconnect();
    public function connect($host = null, $port = 0, $useSsl = false);
    public function bind($username = null, $password = null);
}

class Zend_Ldap_Exception extends Zend_Exception
{
    const LDAP_SUCCESS            = 0x00;
    const LDAP_OPERATIONS_ERROR   = 0x01;
    const LDAP_PROTOCOL_ERROR     = 0x02;
    const LDAP_TIMELIMIT_EXCEEDED = 0x03;
    const LDAP_SIZELIMIT_EXCEEDED = 0x04;
    ... many more error code constants here ...

    public function __construct($ldap = null, $str = null, $code = 0);
    public static function getLdapCode($ldap);
}
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>