Table of Contents

LdapContrib **1**
- Introduction 1
- LDAP questionary 1
- Authentication 2
- User Groups 2
- Membership 2
- Normalization of login, wiki and group names 3
- SSO and LdapContrib 4
- Simple Example 4
- Configuration 5
- Updating the LDAP cache using a cronjob 5
- Implementation documentation 6
  - Foswiki::Contrib::LdapContrib 6
  - Typical usage 6
  - Cache storage format 6
  - writeDebug($msg) 7
  - writeWarning($msg) 7
    - new($session, host=>'...', base=>'...', ...) -> $ldap 7
    - getLdapContrib($session) -> $ldap 7
    - connect($login, $passwd) -> $boolean 8
    - disconnect() 8
    - finish 8
    - checkError($msg) -> $errorCode 8
    - getError() -> $errorMsg 8
    - getAccount($login) -> Net::LDAP::Entry object 8
    - search($filter, %args) -> $msg 8
    - cacheBlob($entry, $attribute, $refresh) -> $pubUrlPath 9
    - initCache() 9
    - refreshCache() -> $boolean 9
    - refreshUsersCache($data) -> $boolean 9
    - resolveWikiNameClashes($data, %wikiNames, %loginNames) -> $integer 9
    - refreshGroups($data) -> $boolean 10
    - cacheUserFromEntry($entry, $data, $wikiNames, $loginNames, $wikiName) -> $boolean 10
    - cacheGroupFromEntry($entry, $data, $groupNames) -> $boolean 10
    - normalizeWikiName($name) -> $string 10
    - normalizeLoginName($name) -> $string 10
    - transliterate($string) -> $string 10
    - getGroupNames($data) -> @array 10
    - isGroup($wikiName, $data) -> $boolean 11
    - getEmails($login, $data) -> @emails 11
    - getLoginOfEmail($email, $data) \@users 11
    - getGroupMembers($groupName, $data) -> \@array 11
    - isGroupMember($loginName, $groupName, $data) -> $boolean 11
    - getWikiNameOfLogin($loginName, $data) -> $wikiName 11
    - getLoginOfWikiName($wikiName, $data) -> $loginName 11
    - getAllWikiNames($data) -> \@array 11
    - getAllLoginNames($data) -> \@array 11
    - getDnOfLogin($loginName, $data) -> $dn 12
    - changePassword($loginName, $newPassword, $oldPassword) -> $boolean 12
    - checkCacheForLoginName($loginName, $data) -> $boolean 12
    - removeGroupFromCache($groupName, $data) -> $boolean 12
    - removeUserFromCache($wikiName, $data) -> $boolean 12
    - addIgnoredUser($loginName, $data) -> \@array 12

Table of Contents

LdapContrib

getAllUnknownUsers($data) -> \@array 12
addIgnoredGroup($groupName, $data) -> \@array 12
getAllUnknownGroups($data) -> \@array 13
checkCacheForLoginName($groupName, $data) -> $boolean 13
getGroup($groupName) -> Net::LDAP::Entry object 13
Foswiki::Users::LdapPassdUser 13
new($session) -> $ldapUser 13
error() -> $errorMsg 13
writeDebug($msg) 14
fetchPass($login) -> $passwd 14
userExists($name) -> $boolean 14
checkPassword($login, $password) -> $boolean 14
readOnly() -> $boolean 14
isManagingEmails() -> $boolean 14
getEmails($login) -> @emails 14
finish() 14
removeUser($user) -> $boolean 15
passwd($user, $newPassword, $newPassword) -> $boolean 15
encrypt($user, $passwordU, $fresh) -> $passwordE 15
setPassword($login, $newPassU, $oldPassU) -> $boolean 15
setEmails($user, @emails) 15
findUserByEmail($email) -> \@users 15
canFetchUsers() -> boolean 16
fetchUsers() -> new Foswiki::ListIterator(\@users) 16
Foswiki::Users::LdapUserMapping 16
new($session) -> $ldapUserMapping 16
finish() 16
writeDebug($msg) 16
addUser ($login, $wikiname, $password, $emails) -> $cUID 16
getLoginName ($cUID) -> $login 16
getWikiName ($cUID) -> wikiname 17
getEmails($cUID) -> @emails 17
userExists($cUID) -> $boolean 17
eachUser () -> listIterator of cUIDs 17
eachGroup () -> listIterator of groupnames 17
getListOfGroups( ) -> @listOfUserObjects 17
eachGroupMember ($groupName) -> listIterator of cUIDs 17
eachMembership ($cUID) -> listIterator of groups this user is in 17
isGroup($user) -> $boolean 17
findUserByEmail($email) -> \@cUIDs 18
findUserByWikiName ($wikiName) -> list of cUIDs associated with that wikiname 18
handlesUser($cUID, $login, $wikiName) -> $boolean 18
login2cUID($loginName, $dontcheck) -> $cUID 18
Dependencies 18
Installation Instructions 18
Contrib Info 19

LdapContrib

Introduction


Powered by
WikiRing Consultants

This package offers basic LDAP services for Foswiki and offers authentication of wiki users by binding to an LDAP server as well as incorporate LDAP user groups into access control.

Optionally, if you need an interface to query your LDAP directory and display the results in a topic install the LdapNgPlugin which will make use of the LdapContrib services. This work is a rewrite of the LdapPlugin by Gerard Hickey while bringing authentication, user management and other LDAP applications onto a common base.

This package downloads all relevant records from your LDAP server into a local cache the first time you use it. This can take a noticeable period of time depending on the size of your LDAP database. This cache will be refreshed on a configurable interval (defaults to once a day). You can also disable automatic refreshing and refresh the LdapContrib's cache manually using the "Refresh Cache" button below. Read the documentation of `MaxCacheAge` in the section "Performance Settings" in configure.

Tip: You can add this button on any page by adding

```
%INCLUDE{"%SYSTEMWEB%.LdapContrib"}%
```

to it.

LDAP questionary

Before you can further configure the LDAP connection you will have to answer a set of basic questions about your LDAP server. These are:

1. What's the host name (or IP address) of the LDAP server (e.g. ldap.my.domain.com)?
2. What port does it listen to LDAP requests (e.g. 389)?
3. Does your LDAP Server use SASL to authenticate connections? If so which authentication mechanism does it use (EXTERNAL, DIGEST-MD5, ...)?
4. Do you have a kind of "proxy" user that the wiki can use to perform the initial connection? You need its DN and credentials. Advice: don't use the LDAP admin account, you only need a simple user that has read access to all of the directory (or the relevant parts); it does not need any write access.
5. What is the "base dn" of the directory (e.g. `dc=my,dc=domain,dc=com`)?
6. What is the common root/branch for all users? For example, Are they all found under `ou=people,dc=my,dc=domain,dc=com` or are they are scattered all over the place?
7. What is the common root/branch where all groups are defined (e.g. ou=group,dc=my,dc=domain,dc=com)?
8. What object class do user records have (e.g. `objectClass=organizationalPerson`)?
9. What object class do group records have (e.g. `objectClass=group`)?
10. Which attribute of a user record should be used to log in (must be unique)?
11. Which attribute(s) of a user record do you want to use to construct a WikiName (used to display them online, pointing to their homepage)?

12. What's the name attribute of a group?
13. Which attribute in a group record defines its members (e.g. member or memberUid)? Note, that if the member attribute of a group is a DN you need to enable "member indirection" (see #Membership).

Collect the answers to these questions either yourself using your favorit LDAP browser, or ask your friendly LDAP admin.

Authentication

To authenticate wiki users using your LDAP server you have to register the LdapPasswdUser class as the so called PasswordManager. This is done by adding the following lines in the `lib/LocalSite.cfg` configuration file (or by using the `configure` tool alternatively):

```
$Foswiki::cfg{PasswordManager} = 'Foswiki::Users::LdapPasswdUser';
```

There is a further option to fallback to the normal authentication mechanism by defining a secondary password manager. This allows you to create native wiki accounts, e.g. a WikiAdmin account and authenticate him without LDAP. Use the following setting to fallback to a htpasswd based authentication.

```
$Foswiki::cfg{Ldap}{SecondaryPasswordManager} = 'Foswiki::Users::HtPasswdUser';
```

So whenever authentication against LDAP fails, this second password manager will be used.

User Groups

LDAP group records can be used in access control lists by registering a UserMappingManager implementation. This is done by adding the following to your `lib/LocalSite.cfg` configuration file (or by using the `configure` tool alternatively):

```
$Foswiki::cfg{UserMappingManager} = 'Foswiki::Users::LdapUserMapping';
```

In addition you can decide if you want to *add* the LDAP groups or use LDAP groups solely. This is controled by the WikiGroupsBackoff flag. If switched on then LDAP groups will be added. If there's a name clash LDAP groups take precedence. If switched off WikiGroups are ignored.

You might decide in not using your LDAP groups but still map login names to WikiNames. Both, LDAP user groups *and* name mapping is done by the UserMappingManager. So to make use of name mapping but *not* its group feature, register the LdapUserMapping implementation for the UserMappingManager but disable the MapGroups setting.

When multiple groups in your LDAP directory clash on the same group name you might actually wish to merge these groups as used online in your wiki. This is done using the `MergeGroups` flag. When disabled, clashing groups are reported as a warning in the server log files when the LDAP cache is refreshed.

Membership

LDAP servers follow different schemata to define "membership". They store the information either using a set of unique ids in the group object (posixGroup) or the full DNs of the user objects (groupOfNames). In the latter case the user objects' unique ids have to be fetched separately based on their distinguished name. This mode has to be switched on using the `MemberIndirection` setting.

The reverse relation, where the *user objects* hold membership information (for example using a `memberOf` attribute) is maintained by some LDAP servers automatically. Those that encode membership this way *only* are not supported by the LdapContrib yet.

Furthermore, user objects may have one *primary* group attribute. This is a simple value that stores the id of a default group that user is member of. This attribute is defined by specifying the `PrimaryGroupAttribute` setting a.

LdapContrib reads membership information as they are stored in the group objects, and may map the member object indirectly to the login name. In addition any "primary group" setting stored in the user objects is consulted as well.

Normalization of login, wiki and group names

LdapContrib reads three kinds of names from your LDAP server and reuses this information as needed. These are the login names - used to log in -, the WikiNames - used to display users online -, and the group names - used in access control lists.

The WikiName can be generated by setting the two parameters `WikiUserNameAttribute` and `NormalizeWikiName`. `WikiUserNameAttribute` can be a comma separated list of LDAP attributes that are then assembled to form the WikiName. If the `NormalizeWikiName` flag is set a couple of extra operations are performed to generate a proper WikiName, i.e. removing illegal characters. Given the setting

```
$Foswiki::cfg{Ldap}{WikiNameAttribute} = 'givenName,sn';
$Foswiki::cfg{Ldap}{NormalizeWikiNames} = 1;
```

The `givenName` and `sn` (surname) LDAP attributes will be fetched and concatenated to form a WikiName, so that "givenName=Hans-Peter,sn=Leuth¿user-Schnarrenberg" will result in the WikiName "HansPeterLeuthaeuserSchnarrenberg". If one of the WikiNameAttributes is 'mail' the `@mydomain.com` part will be removed all together.

The login name can be normalized by enabling the

```
$Foswiki::cfg{Ldap}{NormalizeLoginNames} = 1;
```

setting. However the normalized result will not be forced to be a cammel case WikiName.

Similar to the WikiName of a user, group names can be normalized using

```
$Foswiki::cfg{Ldap}{NormalizeGroupNames} = 1;
```

If a user in your LDAP directory changed his name, e.g. because of a mariage, this use can be mapped to his/her old account using an alias that points back from the old WikiName to the new one. This is done using a setting like this:

```
$Foswiki::cfg{Ldap}{WikiNameAliases} = 'MaryMalone=MaryForrester, ...'
```

The parameter takes a comma separated list of `FromWikiName=ToWikiName`. Whenever this account is still used in an access control list, its rights will be inheritted by the targeted `ToWikiName` account.

Group names can be rewritten using a set of rewrite rules. This is usefull when the names as stored in your LDAP directory don't satisfy your criteria for being displayed online. In conjunction with the MergeGroups flag, separate LDAP groups can be merged onto one group as it is used online in your wiki.

A group rewrite rules has the form

```
'pattern' => 'substitute'
```

consists of a name pattern that has to match the group name to be rewritten and a substitute value that is used to replace the matched pattern. The substitute might contain $1, $2, ... , $5 to insert the first, second, ..., fifth bracket pair in the key pattern. (see perl manual for regular expressions). Example: '(.*)_users' => '$1'

SSO and LdapContrib

First of all, LdapContrib does not provide any SSO solution. Nor does LDAP per se. However, LDAP directories might come with SSO facilities that they provide via kerberos or similar. Unfortunately, nowaday browsers themselves are not kerberized. They depend on talking to the webserver using HTTP means, which then decides which tickets are valid for the remote user by talking to the acutal authority. That is, authentication is implemented using an approriate apache module.

LdapContrib can then be used to complete an LDAP integration by providing the mapping to WikiNames as well as email information and group definitions drawn from the LDAP directory directly.

The remaining problem is that, when a new user has been added to your LDAP directory recently, and he/she then dashes off to sign on to the wiki right away, LdapContrib's cache most probably is outdated. This situation is different from one where users were authenticated by the build-in TemplateLogin mechanism. The user would then not be able to login until the cache has been refreshed manually or automatically.

So when the new user is authenticated using SSO and then hits the wiki, it might fail to compute the proper WikiName. The solution to this problem is to use the LdapApacheLogin login manager as a drop in replacement to the standard ApacheLogin that would have been used in this situation. The LdapApacheLogin will then take care that the remote user is known to the cache and add this single record if it is missing.

Bottomline: whenever you are using apache to authenticate, do use the LdapApacheLogin manager. Or in other words, whenever you configured the wiki to use the standard ApacheLogin manager, and you now install LdapContrib, change it to LdapApacheLogin to assure LdapContrib's cache is up to date.

Simple Example

For the sake of this documentation we assume that users accounts in your database are at leat of the type `posixAccount` and optionally also of type `inetOrgPerson` storing email addresses. Moreover users are stored in a subtree `ou=people` and groups are defined in `ou=group`. Here are some example LDAP records:

```
dn: uid=testuser1,ou=people,dc=my,dc=domain,dc=com
objectClass: inetOrgPerson
objectClass: posixAccount
cn: Test User1
uid: testuser1
uidNumber: 1024
gidNumber: 100
homeDirectory: /home/testuser1
loginShell: /bin/bash
mail: testuser1@my.domain.com

dn: uid=testuser2,ou=people,dc=my,dc=domain,dc=com
```

```
objectClass: inetOrgPerson
objectClass: posixAccount
cn: Test User2
uid: testuser2
uidNumber: 1024
gidNumber: 100
homeDirectory: /home/testuser2
loginShell: /bin/bash
mail: testuser2@my.domain.com
mail: testuser2@gmx.com

# users, Group, nats.informatik.uni-hamburg.de
dn: cn=users,ou=group,dc=my,dc=domain,dc=com
objectClass: posixGroup
cn: users
gidNumber: 100
memberUid: testuser1
memberUid: testuser2
```

Please have a look at your LDAP manual on how to set up an LDAP server and populate it with user account records. Have a look at the Quick-Start Guide on how to install OpenLdap.

Use the following settings for the above example:

```
$Foswiki::cfg{Ldap}{Host} = 'ldap.my.domain.com';
$Foswiki::cfg{Ldap}{Port} = 389;
$Foswiki::cfg{Ldap}{UserBase} = 'ou=people,dc=my,dc=domain,dc=com';
$Foswiki::cfg{Ldap}{LoginFilter} = 'objectClass=posixAccount';
$Foswiki::cfg{Ldap}{LoginAttribute} = 'uid';
$Foswiki::cfg{Ldap}{WikiNameAttribute} = 'cn';
$Foswiki::cfg{Ldap}{NormalizeWikiNames} = 1;
$Foswiki::cfg{Ldap}{GroupBase} = 'ou=group,dc=my,dc=domain,dc=com';
$Foswiki::cfg{Ldap}{GroupFilter} = 'objectClass=posixGroup';
$Foswiki::cfg{Ldap}{GroupAttribute} = 'cn';
$Foswiki::cfg{Ldap}{MemberAttribute} = 'memberUid';
$Foswiki::cfg{Ldap}{MemberIndirection} = 0;
$Foswiki::cfg{Ldap}{MapGroups} = 1;
```

Configuration

The LdapContrib package is configured using a set of variables that need to be added to the `lib/LocalSite.cfg` configuration file. Use the configure tool (at least once) after you installed this package. Have a look at your `lib/LocalSite.cfg` file afterwards. You might also make your changes therein directly to accomodate your installation to your specifc LDAP installation and user accounting. See the documentation within the `configure` tool for an explanation of the various options.

Updating the LDAP cache using a cronjob

In some environments, updating the internal LDAP cache of the LdapContrib might take considerable time. The intervals the cache data is thought to be "expired" is configured using the `MaxCacheAge` setting. This setting defaults to updating the cache every 24 hours. The refresh procedure will then be triggered by the first request that hits the site when this period expired.

To remove this burden from the "first visitor in the morning", the automatic refresh procedure can be disabled by setting

```
$Foswiki::cfg{Ldap}{MaxCacheAge} = 0;
```

This means that the age of the cached data will not be checked *automatically* anymore. The responsibility that the data is updated is now up to you, that is you have to update the cache *explicitly*. This can be done by either hitting the red "Refresh Cache" button above, or by setting up an appropriate cronjob on the machine running your wiki server.

To trigger an explicit update of the cache on 5 past midnight every day use a cronjob similar to:

```
5 0 * * * cd <wiki-install-path>/bin && ./view refreshldap=on Main/WebHome >/dev/null
```

This will call the engine on the commandline and provide the necessary query parameters so that the LdapContrib will force an update of the cache data.

Implementation documentation

Foswiki::Contrib::LdapContrib

General LDAP services module. This class encapsulates the platform-specific means to integrate an LDAP directory service. Used by Foswiki::Users::LdapPasswdUser for authentication, Foswiki::Users::LdapUserMapping for group definitions and to interface general query services.

Typical usage

```
my $ldap = new Foswiki::Contrib::LdapContrib;

my $result = $ldap->search(filter=>'mail=*@gmx*');
my $errorMsg = $ldap->getError();

my $count = $result->count();

my @entries = $result->sorted('sn');
my $entry = $result->entry(0);

my $value = $entry->get_value('cn');
my @emails = $entry->get_value('mail');
```

Cache storage format

The cache stores a series of key-value pairs in a DB_File. The following keys are used:

- WIKINAMES - list of all wikiNames
- LOGINNAMES - list of all loginNames
- GROUPS - list of all groups
- UNKWNUSERS - list of all usernames that could not be found in LDAP (to avoid future LDAP lookups in case caching is OFF)
- UNKWNGROUPS - list of all group names that could not be found in LDAP (to avoid future LDAP lookups in case caching is OFF)
- GROUPS::$groupName - list of all loginNames in group groupName (membership)

- GROUP2UNCACHEDMEMBERSDN?::$groupName - list of all DNs (when in memberIndirection mode) that could not be resolved to a user or group existing in the cache when $groupName was retreived from LDAP
- EMAIL2U?::$emailAddr - stores the loginName of an emailAddr
- U2EMAIL?::$loginName - stores the emailAddr of a loginName
- U2W?::$loginName - stores the wikiName of a loginName
- W2U?::$wikiName - stores the loginName of a wikiName
- DN2U?::$dn - stores the loginName of a distinguishedName
- U2DN?::$loginName - stores the distinguishedName of a loginName

writeDebug($msg)

Static Method to write a debug messages.

writeWarning($msg)

Static Method to write a warning messages.

new($session, host=>'...', base=>'...', ...) -> $ldap

Construct a new Foswiki::Contrib::LdapContrib object

Possible options are:

- host: ip address (or hostname)
- base: the base DN to use in searches
- port: port address used when binding to the LDAP server
- version: protocol version
- userBase: sub-tree DN of user accounts
- groupBase: sub-tree DN of group definitions
- loginAttribute: user login name attribute
- loginFilter: filter to be used to find login accounts
- groupAttribute: the group name attribute
- groupFilter: filter to be used to find groups
- memberAttribute: the attribute that should be used to collect group members
- innerGroupAttribute: the attribute that should be used to collect inner groups of a group
- bindDN: the dn to use when binding to the LDAP server
- bindPassword: the password used when binding to the LDAP server

Options not passed to the constructor are taken from the global settings in `lib/LocalSite.cfg`.

getLdapContrib($session) -> $ldap

Returns a standard singleton Foswiki::Contrib::LdapContrib object based on the site-wide configuration.

connect($login, $passwd) -> $boolean

Connect to LDAP server. If a $login name and a $passwd is given then a bind is done. Otherwise the communication is anonymous. You don't have to connect() explicitely by calling this method. The methods below will do that automatically when needed.

disconnect()

Unbind the LDAP object from the server. This method can be used to force a reconnect and possibly rebind as a different user.

finish

finalize this ldap object.

checkError($msg) -> $errorCode

Private method to check a Net::LDAP::Message object for an error, sets $ldap->{error} and returns the ldap error code. This method is called internally whenever a message object is returned by the server. Use $ldap->getError() to return the actual error message.

getError() -> $errorMsg

Returns the error message of the last LDAP action or undef it no error occured.

getAccount($login) -> Net::LDAP::Entry object

Fetches an account entry from the database and returns a Net::LDAP::Entry object on success and undef otherwise. Note, the login name is match against the attribute defined in $ldap->{loginAttribute}. Account records are search using $ldap->{loginFilter} in the subtree defined by $ldap->{userBase}.

search($filter, %args) -> $msg

Returns an Net::LDAP::Search object for the given query on success and undef otherwise. If $args{base} is not defined $ldap->{base} is used. If $args{scope} is not defined 'sub' is used (searching down the subtree under $args{base}. If no $args{limit} is set all matching records are returned. The $attrs is a reference to an array of all those attributes that matching entries should contain. If no $args{attrs} is defined all attributes are returned.

If undef is returned as an error occured use $ldap->getError() to get the cleartext message of this search() operation.

Typical usage:

```
my $result = $ldap->search(filter=>'uid=TestUser');
```

cacheBlob($entry, $attribute, $refresh) -> $pubUrlPath

Takes an Net::LDAP::Entry and an $attribute name, and stores its value into a file. Returns the pubUrlPath to it. This can be used to store binary large objects like images (jpegPhotos) into the filesystem accessible to the httpd which can serve it in return to the client browser.

Filenames containing the blobs are named using a hash value that is generated using its DN and the actual attribute name whose value is extracted from the database. If the blob already exists in the cache it is *not* extracted once again except the $refresh parameter is defined.

Typical usage:

```
my $blobUrlPath = $ldap->cacheBlob($entry, $attr);
```

initCache()

loads/connects to the LDAP cache

refreshCache() -> $boolean

download all relevant records from the LDAP server and store it into a database

refreshUsersCache($data) -> $boolean

download all user records from the LDAP server and cache it into the given hash reference

returns true if new records have been loaded

resolveWikiNameClashes($data, %wikiNames, %loginNames) -> $integer

if there have been name clashes during cacheIserFromEntry() those entry records have not yet been added to the cache. They are kept until all clashes have been found and a deterministic renaming scheme can be applied. Clashed WikiNames will be enumerated - WikiName1?, WikiName2?, WikiName3? etc - and finally added to the database. The renaming is kept stable by sorting the dn entry of all clashed entries.

returns the number of additional entries that have been cached

refreshGroups($data) -> $boolean

download all group records from the LDAP server

returns true if new records have been loaded

cacheUserFromEntry($entry, $data, $wikiNames, $loginNames, $wikiName) -> $boolean

store a user LDAP::Entry to our internal cache

If the $wikiName parameter is given explicitly then this will be the name under which this record will be cached.

returns true if new records have been created

cacheGroupFromEntry($entry, $data, $groupNames) -> $boolean

store a group LDAP::Entry to our internal cache

returns true if new records have been created

normalizeWikiName($name) -> $string

normalizes a string to form a proper WikiName

normalizeLoginName($name) -> $string

normalizes a string to form a proper login

transliterate($string) -> $string

transliterate some essential utf8 chars to a common replacement in latin1 encoding. the list above is not exhaustive.

use http://www.ltg.ed.ac.uk/~richard/utf-8.html to add more recodings

getGroupNames($data) -> @array

Returns a list of known group names.

isGroup($wikiName, $data) -> $boolean

check if a given user is an ldap group actually

getEmails($login, $data) -> @emails

fetch emails from LDAP

getLoginOfEmail($email, $data) \@users

get all users matching a given email address

getGroupMembers($groupName, $data) -> \@array

isGroupMember($loginName, $groupName, $data) -> $boolean

check if a given user is member of an ldap group

getWikiNameOfLogin($loginName, $data) -> $wikiName

returns the wikiName of a loginName or undef if it does not exist

getLoginOfWikiName($wikiName, $data) -> $loginName

returns the loginNAme of a wikiName or undef if it does not exist

getAllWikiNames($data) -> \@array

returns a list of all known wikiNames

getAllLoginNames($data) -> \@array

returns a list of all known loginNames

getDnOfLogin($loginName, $data) -> $dn

returns the Distinguished Name of the LDAP record of the given name

changePassword($loginName, $newPassword, $oldPassword) -> $boolean

checkCacheForLoginName($loginName, $data) -> $boolean

grant that the current loginName is cached. If not, it will download the LDAP record for this specific user and update the LDAP cache with this single record.

This happens when the user is authenticated externally, e.g. using apache's mod_authz_ldap or some other SSO, and the internal cache is not yet updated. It is completely updated regularly on a specific time interval (default every 24h). See the LdapContrib settings.

removeGroupFromCache($groupName, $data) -> $boolean

Remove a group from the cache

removeUserFromCache($wikiName, $data) -> $boolean

removes a wikiName from the cache

addIgnoredUser($loginName, $data) -> \@array

Insert a new user in the list of unknown users that should not be lookedup in LDAP

getAllUnknownUsers($data) -> \@array

returns a list of all unknown users that should not be relookedup in LDAP

addIgnoredGroup($groupName, $data) -> \@array

Insert a new group in the list of unknown groups that should not be lookedup in LDAP

getAllUnknownGroups($data) -> \@array

returns a list of all unknown groups that should not be relookedup in LDAP

checkCacheForLoginName($groupName, $data) -> $boolean

grant that the current groupName is cached. If not, it will download the LDAP record for this specific group and its subgroups and update the LDAP cache with the retreived records.

This happens when the precache mode is off. See the LdapContrib settings.

getGroup($groupName) -> Net::LDAP::Entry object

Fetches a group entry from the database and returns a Net::LDAP::Entry object on success and undef otherwise. Note, the group name is match against the attribute defined in $ldap->{groupAttribute}. Account records are search using $ldap->{groupFilter} in the subtree defined by $ldap->{groupBase}.

Foswiki::Users::LdapPassdUser

Password manager that uses Net::LDAP to manage users and passwords.

Subclass of `Foswiki::Users::Password`.

This class does not grant any write access to the ldap server for security reasons. So you need to use your ldap tools to create user accounts.

Configuration: add the following variables to your LocalSite.cfg

- $Foswiki::cfg{Ldap}{server} = <ldap-server uri>, defaults to localhost
- $Foswiki::cfg{Ldap}{base} = <base dn> subtree that holds the user accounts e.g. ou=people,dc=your,dc=domain,dc=com

new($session) -> $ldapUser

Takes a session object, creates an LdapContrib object used to delegate LDAP calls and returns a new Foswiki::User::LdapPasswd object

error() -> $errorMsg

return the last error during LDAP operations

writeDebug($msg)

Static method to write a debug messages.

fetchPass($login) -> $passwd

this method is used most of the time to detect if a given login user is known to the database. the concrete (encrypted) password is of no interest: so better use userExists() for that

userExists($name) -> $boolean

returns true if the login or wikiname exists in the database; that's performing better than fetching the password and then see what comes out of this

checkPassword($login, $password) -> $boolean

check passwd by binding to the ldap server

readOnly() -> $boolean

we can change passwords, so return false

isManagingEmails() -> $boolean

we are managing emails, but don't allow setting emails. alas the core does not distinguish this case, e.g. by using readOnly()

getEmails($login) -> @emails

emails might be stored in the ldap account as well if the record is of type possixAccount and inetOrgPerson. if this is not the case we fallback to twiki's default behavior

finish()

Complete processing after the client's HTTP request has been responded. i.e. destroy the ldap object.

removeUser($user) -> $boolean

LDAP users can't be removed from within the engine. So this will call the deleteUser interface of the secondary password manager only

Returns 1 on success, undef on failure.

passwd($user, $newPassword, $newPassword) -> $boolean

TODO: API missmatch

This method can only change the LDAP password. It can not add the user to the LDAP directory. To change the password the old password must always be correct. There's no mode to force the change irrespective of the existing password.

In any other case the secondary password manager gets the job.

encrypt($user, $passwordU, $fresh) -> $passwordE

LDAP can't encrypt passwords. But maybe the secondary password manager can.

setPassword($login, $newPassU, $oldPassU) -> $boolean

If the $oldPassU matches matches the user's password, then it will replace it with $newPassU.

If $oldPassU is not correct and not 1, will return 0.

If $oldPassU is 1, will force the change irrespective of the existing password, adding the user if necessary.

Otherwise returns 1 on success, undef on failure.

setEmails($user, @emails)

Set the email address(es) for the given username. The engine can't set the email stored in LDAP. But may be the secondary password manager can.

findUserByEmail($email) -> \@users

- `$email` - email address to look up

Return a list of user objects for the users that have this email registered with the password manager. This will concatenate the result list of the LDAP manager with the secondary password manager

canFetchUsers() -> boolean

returns true, as we can fetch users

fetchUsers() -> new Foswiki::ListIterator(\@users)

returns a FoswikiIterator? of loginnames

Foswiki::Users::LdapUserMapping

This class allows to use user names and groups stored in an LDAP database inside Foswiki in a transparent way. This replaces Foswiki's native way to represent users and groups using topics with according LDAP records.

new($session) -> $ldapUserMapping

create a new Foswiki::Users::LdapUserMapping object and constructs an LdapContrib object to delegate LDAP services to.

finish()

Complete processing after the client's HTTP request has been responded to. I.e. it disconnects the LDAP database connection.

writeDebug($msg)

Static method to write a debug messages.

addUser ($login, $wikiname, $password, $emails) -> $cUID

overrides and thus disables the SUPER method

getLoginName ($cUID) -> $login

Converts an internal cUID to that user's login (undef on failure)

getWikiName ($cUID) -> wikiname

Maps a canonical user name to a wikiname

getEmails($cUID) -> @emails

emails might be stored in the ldap account as well if the record is of type possixAccount and inetOrgPerson. if this is not the case we fallback to the default behavior

userExists($cUID) -> $boolean

Determines if the user already exists or not.

eachUser () -> listIterator of cUIDs

returns a list iterator for all known users

eachGroup () -> listIterator of groupnames

returns a list iterator for all known groups

getListOfGroups( ) -> @listOfUserObjects

Get a list of groups defined in the LDAP database. If `nativeGroupsBackoff` is defined the set of LDAP and native groups will merged whereas LDAP groups have precedence in case of a name clash.

eachGroupMember ($groupName) -> listIterator of cUIDs

returns a list iterator for all groups members

eachMembership ($cUID) -> listIterator of groups this user is in

returns a list iterator for all groups a user is in.

isGroup($user) -> $boolean

Establish if a user object refers to a user group or not. This returns true for the SuperAdminGroup or the known LDAP groups. Finally, if `nativeGroupsBackoff` is set the native mechanism are used to check if

$user is a group

findUserByEmail($email) -> \@cUIDs

- `$email` - email address to look up

Return a list of canonical user names for the users that have this email registered with the password manager or the user mapping manager.

findUserByWikiName ($wikiName) -> list of cUIDs associated with that wikiname

See baseclass for documentation

handlesUser($cUID, $login, $wikiName) -> $boolean

Called by the Foswiki::Users object to determine which loaded mapping to use for a given user.

The user can be identified by any of $cUID, $login or $wikiName. Any of these parameters may be undef, and they should be tested in order; cUID first, then login, then wikiName.

login2cUID($loginName, $dontcheck) -> $cUID

Convert a login name to the corresponding canonical user name. The canonical name can be any string of 7-bit alphanumeric and underscore characters, and must correspond 1:1 to the login name. (undef on failure)

(if dontcheck is true, return a cUID for a nonexistant user too. This is used for registration)

Dependencies

Name	Version	Description
Authen::SASL	>=2.00	Optional
DB_File	>=1.00	Required
Digest::MD5	>=2.36	Required
Net::LDAP	>=0.33	Required
IO::Socket::SSL	>=1.0	Optional
Unicode::MapUTF8	>=1.11	Required

Installation Instructions

You do not need to install anything in the browser to use this extension. The following instructions are for the administrator who installs the extension on the server.

Open configure, and open the "Extensions" section. Use "Find More Extensions" to get a list of available extensions. Select "Install".

If you have any problems, or if the extension isn't available in `configure`, then you can still install manually from the command-line. See http://foswiki.org/Support/ManuallyInstallingExtensions for more help.

- Read the the above documentation, i.e. the Configuration section.
- Use configure to set the LDAP settings.

Contrib Info

This work was partly sponsored by

- Spanlink Communications
- Trivadis
- IBM
- Deutsche Flugsicherung
- Testo

Author:	Michael Daum
Copyright ©:	2006-2009 Michael Daum http://michaeldaumconsulting.com
License:	GPL (GNU General Public License)
Release:	v4.00
Version:	5565 (2009-11-17)
Change History:	
17 Nov 2009:	incremental cache updates for *big* ldap directories and inner groups feature (Cyril Bousquet, IBM); added scope parameter searching for users and groups; added rewrite rules for wiki names; added name clash resolution for wiki names; fixed race condition in cache upate leading to file corruption; cleanup of internal api to properly deal with interim caching structures
25 May 2009:	extended transliteration of utf8 chars to polish chars (Bartosz Dziuda)
09 Apr 2009:	added RewriteGroups and MergeGroups feature
08 Apr 2009:	fixing nested ldap groups
02 Mar 2009:	prevent error when called in the middle of the Foswiki constructor
27 Feb 2009:	fixing use of uninitialized value in user mapper; optimized check for existing user to respect the exclude configuration setting; renamed LdapPassword to LdapPasswdUser again to please configure; transliteration of umlaute (this seems to happen occasionally during svn ci/co or whatever)
11 Feb 2009:	allow utf8 chars in passwords; all strings from LDAP are tainted, even if it comes from mod_ldap via remote_user; only use LDAP query as a last resort to check if a user exists; prevent explicitly excluded login names to be added to the cache
20 Jan 2009:	fixed geting emails for groups and login names
19 Dec 2008:	fixed group mapping under MapGroup=off
05 Dec 2008:	fixed group mapping
04 Dec 2008:	fixed getting email info
06 Oct 2008:	dropped support for TWiki < 4.2.3; added support TLS encryption (by Wolfgang Karall)
12 Jun 2008:	added workarounds to use LDAP and MailInContrib on TWiki-4.2.0

25 May 2008:	added alias feature, fixed normalization error, fixed cache update issue added login manager for 4.2
05 May 2008:	implemented WikiNamesAliases
14 Feb 2008:	allow to disable cache aging setting MaxCacheAge to zero
01 Feb 2008:	distinguish groups clashing with user names by appending a suffix
30 Jan 2008:	first beta towards TWiki-4.2
07 Jan 2008:	fixed initializing the cache
21 Dec 2007:	added LdapApacheLogin, made updating the cache quasi atomic
22 Nov 2007:	fixed recognition of WikiGroups in a mixed setting
05 Oct 2007:	enabled native user registration using the secondary password manager; added support to change a user's LDAP password from within TWiki; added patch for `TWiki.pm` that backports some of the fixes from TWiki-4.2 to TWiki-4.1.2
05 Sep 2007:	added SASL support, added normalization of login and group names, added secondary password manager
31 Aug 2007:	rewrite of the cache
08 June 2007:	don't use the store object during TWiki's destructore; don't lookup login names of groups
04 June 2007:	don't be casesensitive for login names; fixed several utf8 issues; fixed crash when no groups where found; caching mapping privately; added MaxCacheAge; added support for nested LDAP groups
30 Apr 2007:	fixed return value on illegal lookup calls
24 Apr 2007:	be robust against the lookup-API being called with the wrong parameters; added `Debug` flag; fixed/improved group loading; deprecating `BasePasswd` in favor of `UserBase`; deprecating `BaseGroup` in favor of `GroupBase`
04 Apr 2007:	fixed group mapping on >4.1.2; renamed BasePasswd config parameter to UserBase; renamed BaseGroup config parameter to GroupBase; working around broken `configure` in 4.1.x
12 Jan 2007:	enhanced normalization of WikiNames so that they are proper WikiWords; WikiNames can be constructed from a list of LDAP attributes now
18 Dec 2006:	various performance improvements; fixed usage of `limit` argument; renamed configuration option "WikiNameRemoveWhiteSpace" to "NormalizeWikiName"; support for large databases using paged LDAP search results; new configuration option "Exclude" to exclude standard TWiki user accounts, e.g. RegistrationAgent, from being looked up in LDAP; added support for faster API implementing `isMemberOf`; added Config.spec file to integrate the LadpContrib into Twiki's "configure" tool; added support for WikiNames derived from mail attributes
03 Nov 2006:	fixed binding to the server by first searching the full dn instead of assuming a fixed one (issue found by Cederic Weber); added new feature MapGroup to be able to switch off group mapping and have ; login-to-wikiname conversion only
02 Aug 2006:	added a user accounts in memory cache
19 July 2006:	public release
24 May 2006:	api adjustments, improved wikiname generation
28 Apr 2006:	Initial version
Home:	Foswiki:Extensions/LdapContrib
Support:	Foswiki:Support/LdapContrib

Edit | Attach | Print version | History: %REVISIONS% | Backlinks | Raw View | More topic actions

Topic revision: r2 - 19 Oct 2009 - 19:50:27 - ProjectContributor

- . System

- Log In

- **Toolbox**
- Users
- Groups
- Index
- Search
- Changes
- Notifications
- RSS Feed
- Statistics
- Preferences

- **User Reference**
- BeginnersStartHere
- TextFormattingRules
- Macros
- FormattedSearch
- QuerySearch
- DocumentGraphics
- SkinBrowser
- InstalledPlugins

- **Admin Maintenance**
- Reference Manual
- AdminToolsCategory
- InterWikis
- ManagingWebs
- SiteTools
- DefaultPreferences
- WebPreferences

- **Categories**
- Admin Documentation
- Admin Tools
- Developer Doc
- User Documentation
- User Tools

- **Webs**
- Public
- System

Copyright © by the contributing authors. All material on this site is the property of the contributing authors.
Ideas, requests, problems regarding Wiki? Send feedback