mod_shared_roster_ldap 0.5.3

Shared Roster LDAP Documentation

Marcin Owsiany

Contents

1 Introduction **5**

1.1 History 5

1.2 How does `mod_shared_roster_ldap` work 6

1.3 Shameless plug 6

2 Installing `mod_shared_roster_ldap` **7**

2.1 Installing with `ejabberd` from source 7

2.2 Installing with an `ejabberd` binary package 7

3 Configuring `mod_shared_roster_ldap` **9**

3.1 Configuration parameters 9

3.1.1 Filters 9

3.1.2 Attributes 10

3.1.3 Control parameters 11

3.1.4 Connection parameters 12

3.2 Module startup 12

3.3 Retrieving the roster 12

3.4 Configuration examples 13

3.4.1 Flat DIT 13

3.4.2 Deep DIT 15

A Release Notes **19**

B Copyright Information **23**

Chapter 1

Introduction

`ejabberd` is a free and open source instant messaging server written in Erlang/OTP[1].

`mod_shared_roster_ldap` is a module for `ejabberd` which lets the server administrator automatically populate users' rosters (contact lists) with entries based on users and groups defined in an LDAP-based directory.

1.1 History

The module was initially written in 2005 by Alexey Shchepin (mailto:alexey@sevcom.net).

It was subsequently changed by Realloc (mailto:realloc@realloc.spb.ru) to make it Active Directory friendly and more usable. This developer has produced a russian-language web page about AD integration[2].

The module has spent some time posted on its contribution page[3] where it has received fixes and minor improvements, however it was not actively developed nor properly maintained.

The most often requested part that was missing was comprehensive documentation. This document attempts to provide it. It was written by incorporating my own interpretation of the code and various descriptions contributed by other people on the `ejabberd` forums, e.g.:

- http://www.ejabberd.im/node/1317
- http://www.ejabberd.im/node/3711#comment-54820

This documentation attempts to be comprehensive and correct. However since it was written by analyzing the code, it may not follow the code author's exact intentions. Corrections and suggestions are welcome.

[1] http://www.erlang.org/
[2] http://realloc.spb.ru/share/ejabberd112ad.html
[3] http://www.ejabberd.im/mod_shared_roster_ldap

This document, and `mod_shared_roster_ldap` code is maintained at the ejabberd-msrl project page[4] on Alioth. The goal of the project is to provide a place for proper maintenance (with bug tracker, revision control, etc), where the state of this module documentation, featureset and performance can be improved.

1.2 How does `mod_shared_roster_ldap` work

The module does its job by a set of hooks, which it registers in the server on startup. Those hooks intercept the information flowing between a user and `ejabberd` and amend it with data retrieved from LDAP in such way as to provide the user with a permanent set of (additional) "virtual" entries in her roster.

"Virtual" in this context means that the module does not modify the rosters stored by the `mod_roster` module. Instead it "overlays" some additional entries on top of the ones maintained by the user herself, every time the user's client retrieves the roster when connecting to `ejabberd`. This also means that the user cannot remove a `mod_shared_roster_ldap` entry from their roster permanently — it will be included in the roster on next reconnection.

1.3 Shameless plug

The LDAP graph pictures in section 3.4 were created with ldif2dot.[5]

[4] https://alioth.debian.org/projects/ejabberd-msrl/
[5] http://marcin.owsiany.pl/ldif2dot-page

Chapter 2

Installing mod_shared_roster_ldap

2.1 Installing with ejabberd from source

If you are installing ejabberd from source, then simply copying the mod_shared_roster_ldap.erl, mod_shared_roster_ldap.hrl and mod_shared_roster_ldap_helpers.erl files into the src/ directory before running make will cause the modules to be compiled and installed with the rest of ejabberd.

2.2 Installing with an ejabberd binary package

If ejabberd has been installed from a binary package (or using the binary installer), you will need to build and install the module by yourself. Here are some instructions:

1. you need an Erlang runtime and compiler installation, they probably come together — check whether you have the erl and erlc commands. You should probably use the same (or close enough) erlang compiler version as the one which was used to compile your binary ejabberd installation.

2. you also need an unpacked *source* package of ejabberd (strictly speaking only the *.hrl headers are needed) for the same version as you binary ejabberd installation,[1]

3. copy the files mod_shared_roster_ldap_helpers.erl, mod_shared_roster_ldap.hrl and mod_shared_roster_ldap.erl, into the src/ subdirectory of ejabberd source tree

4. compile the modules by running the following in a terminal:[2]

[1] If you run a Debian-based system, you should be able to get that easily with just apt-get install dpkg-dev ; apt-get source ejabberd

[2] You need to have the compiler command erlc in your execution PATH variable, or specify the full path to erlc. In Windows it will be something like "c:\Program Files\Erl5.6.5\bin\erlc.exe"

```
erlc mod_shared_roster_ldap.erl
erlc mod_shared_roster_ldap_helpers.erl
```

5. copy the resulting `mod_shared_roster_ldap.beam` and `mod_shared_roster_ldap_helpers.beam` to the `ejabberd` ebin directory[3]
6. restart `ejabberd` to let it load the module,

[3] this will be something like `/usr/lib/ejabberd/ebin` or `lib/ejabberd-`*your-version*`/ebin/` depending on your system.

Chapter 3

Configuring mod_shared_roster_ldap

3.1 Configuration parameters

The module accepts the following configuration parameters. Some of them, if unspecified for `mod_shared_roster_ldap`, default to the values specified for the top level of configuration. This lets you avoid specifying, for example, the bind password, in multiple places.

3.1.1 Filters

These parameters specify LDAP filters used to query for shared roster information. All of them are run against the `ldap_base`.

`ldap_rfilter` So called "Roster Filter". Used to find names of all "shared roster" groups. See also the `ldap_groupattr` parameter. If unspecified, defaults to the top-level parameter of the same name. You *have to* specify it in some place in the configuration, there is no default.

`ldap_ufilter` "User Filter" – used for retrieving the human-readable name of roster entries (usually full names of people in the roster). See also the parameters `ldap_userdesc` and `ldap_useruid`. If unspecified, defaults to the top-level parameter of the same name. If that one also is unspecified, then the filter is assembled from values of other parameters as follows (`[ldap_SOMETHING]` is used to mean "the value of the configuration parameter `ldap_SOMETHING`"):

```
(&(&([ldap_memberattr]=[ldap_memberattr_format])([ldap_groupattr]=%g))[ldap_filter])
```

Subsequently `%u` and `%g` are replaced with a `*`. This means that given the defaults, the filter sent to the LDAP server is would be `(&(memberUid=*)(cn=*))`. If however the

`ldap_memberattr_format` is something like `uid=%u,ou=People,o=org`, then the filter will be `(&(memberUid=uid=*,ou=People,o=org)(cn=*))`.

`ldap_gfilter` "Group Filter" – used when retrieving human-readable name (a.k.a. "Display Name") and the members of a group. See also the parameters `ldap_groupattr`, `ldap_groupdesc` and `ldap_memberattr`. If unspecified, defaults to the top-level parameter of the same name. If that one also is unspecified, then the filter is constructed exactly in the same way as `User Filter`.

`ldap_filter` Additional filter which is AND-ed together with `User Filter` and `Group Filter`. If unspecified, defaults to the top-level parameter of the same name. If that one is also unspecified, then no additional filter is merged with the other filters.

Note that you will probably need to manually define the `User` and `Group Filter`s (since the auto-assembled ones will not work) if:

- your `ldap_memberattr_format` is anything other than a simple `%u`,
- **and** the attribute specified with `ldap_memberattr` does not support substring matches.

An example where it is the case is OpenLDAP and `(unique)MemberName` attribute from the `groupOf(Unique)Names` objectClass. A symptom of this problem is that you will see messages such as the following in your `slapd.log`:

```
get_filter: unknown filter type=130
filter="(&(?=undefined)(?=undefined)(something=else))"
```

3.1.2 Attributes

These parameters specify the names of the attributes which hold interesting data in the entries returned by running filters specified in section 3.1.1.

`ldap_groupattr` The name of the attribute that holds the group name, and that is used to differentiate between them. Retrieved from results of the "Roster Filter" and "Group Filter". Defaults to `cn`.

`ldap_groupdesc` The name of the attribute which holds the human-readable group name in the objects you use to represent groups. Retrieved from results of the "Group Filter". Defaults to whatever `ldap_groupattr` is set.

`ldap_memberattr` The name of the attribute which holds the IDs of the members of a group. Retrieved from results of the "Group Filter". Defaults to `memberUid`.

The name of the attribute differs depending on the `objectClass` you use for your group objects, for example:

`posixGroup` → `memberUid`

`groupOfNames` → `member`

`groupOfUniqueNames` → `uniqueMember`

`ldap_userdesc` The name of the attribute which holds the human-readable user name. Retrieved from results of the "User Filter". Defaults to `cn`.

`ldap_useruid` The name of the attribute which holds the ID of a roster item. Value of this attribute in the roster item objects needs to match the ID retrieved from the `ldap_memberattr` attribute of a group object. Retrieved from results of the "User Filter". Defaults to `cn`.

3.1.3 Control parameters

These paramters control the behaviour of the module.

`ldap_memberattr_format` A globbing format for extracting user ID from the value of the attribute named by `ldap_memberattr`. Defaults to `%u`, which means that the whole value is the member ID. If you change it to something different, you may also need to specify the User and Group Filters manually — see section 3.1.1.

`ldap_memberattr_format_re` A regex for extracting user ID from the value of the attribute named by `ldap_memberattr`.

An example value `"CN=(\\w*),(OU=.*,)*DC=company,DC=com"` works for user IDs such as the following:

- `CN=Romeo,OU=Montague,DC=company,DC=com`
- `CN=Abram,OU=Servants,OU=Montague,DC=company,DC=com`
- `CN=Juliet,OU=Capulet,DC=company,DC=com`
- `CN=Peter,OU=Servants,OU=Capulet,DC=company,DC=com`

In case:

- the option is unset,
- or the `re` module in unavailable in the current Erlang environment,
- or the regular expression does not compile,

then instead of a regular expression, a simple format specified by `ldap_memberattr_format` is used. Also, in the last two cases an error message is logged during the module initialization.

Also, note that in all cases `ldap_memberattr_format` (and *not* the regex version) is used for constructing the default "User/Group Filter" — see section 3.1.1.

`ldap_auth_check` Whether the module should check (via the ejabberd authentication subsystem) for existence of each user in the shared LDAP roster. See section 3.3 form more information. Set to `off` if you want to disable the check. Defaults to `on`.

`ldap_user_cache_validity` Number of seconds for which the cache for roster item full names is considered fresh after retrieval. 300 by default. See section 3.3 on how it is used during roster retrieval.

`ldap_group_cache_validity` Number of seconds for which the cache for group membership is considered fresh after retrieval. 300 by default. See section 3.3 on how it is used during roster retrieval.

3.1.4 Connection parameters

The module also accepts the following parameters, all of which default to the top-level parameter of the same name, if unspecified. See the `ejabberd` User Guide chapter 3.2.5 LDAP Configuration[1] for more information about them.

`ldap_servers` List of LDAP server hostnames to connect to.

`ldap_port` Port to use for LDAP connections. The default is determined based on the TLS settings.

`ldap_base` Search base DN — the module will look for entries under this element.

`ldap_rootdn` The "bind DN" to use.

`ldap_password` The bind password.

`ldap_encrypt` The encryption protocol to use.

`ldap_tls_verify` The encryption protocol to use.

3.2 Module startup

When the module is loaded, `ejabberd` spawns a separate module instance for each hosted domain. Each instance performs the following actions on startup:

1. reads and parses the configuration options,
2. prepares the default filter strings which will be used during its operation, unless they were specified explicitly in the configuration (see section 3.1.1).
3. registers callbacks with some `ejabberd` hooks, that will cause it to be invoked at various points in roster lifecycle,
4. spawns a persistent connection to the LDAP server,
5. starts listening for requests — see the following sections for information on how it serves them

3.3 Retrieving the roster

When the module is called to retrieve the shared roster for a user, the following algorithm is used:

1. A list of names of groups to display is created: the `Roster Filter` is run against the base DN, retrieving the values of the attribute named by `ldap_groupattr`.

[1] http://www.process-one.net/en/ejabberd/guide_en#ldap

2. Unless the group cache is fresh (see the `ldap_group_cache_validity` option), it is refreshed:
   (a) Information for all groups is retrieved using a single query: the `Group Filter` is run against the Base DN, retrieving the values of attributes named by `ldap_groupattr` (group ID), `ldap_groupdesc` (group "Display Name") and `ldap_memberattr` (IDs of group members).
   (b) group "Display Name", read from the attribute named by `ldap_groupdesc`, is stored in the cache for the given group
   (c) the following processing takes place for each retrieved value of attribute named by `ldap_memberattr`:
      i. the user ID part of it is extracted using `ldap_memberattr_format(_re)`,
      ii. then (unless `ldap_auth_check` is set to `off`) for each found user ID, the module checks (using the `ejabberd` authentication subsystem) whether such user exists in the given virtual host. It is skipped if the check is enabled and fails.
      This step is here for historical reasons. If you have a tidy DIT and properly defined "Roster Filter" and "Group Filter", it is safe to disable it by setting `ldap_auth_check` to `off` — it will speed up the roster retrieval.
      iii. the user ID is stored in the list of members in the cache for the given group
3. For each item (group name) in the list of groups retrieved in step 1:
   (a) the display name of a shared roster group is retrieved from the group cache
   (b) for each IDs of users which belong to the group, retrieved from the group cache:
      i. the ID is skipped if it's the same as the one for which we are retrieving the roster. This is so that the user does not have himself in the roster.
      ii. the display name of a shared roster user is retrieved:
         A. first, unless the user name cache is fresh (see the `ldap_user_cache_validity` option), it is refreshed by running the `User Filter`, against the Base DN, retrieving the values of attributes named by `ldap_useruid` and `ldap_userdesc`.
         B. then, the display name for the given user ID is retrieved from the user name cache.

3.4 Configuration examples

Since there are many possible DIT[2] layouts, it will probably be easiest to understand how to configure the module by looking at an example for a given DIT (or one resembling it).

3.4.1 Flat DIT

This seems to be the kind of DIT for which this module was initially designed. Basically there are just user objects, and group membership is stored in an attribute individually for each user. For example in a layout shown in figure 3.1, the group of each user is stored in its `ou` attribute.

Such layout has a few downsides, including:

[2] http://en.wikipedia.org/wiki/Directory_Information_Tree

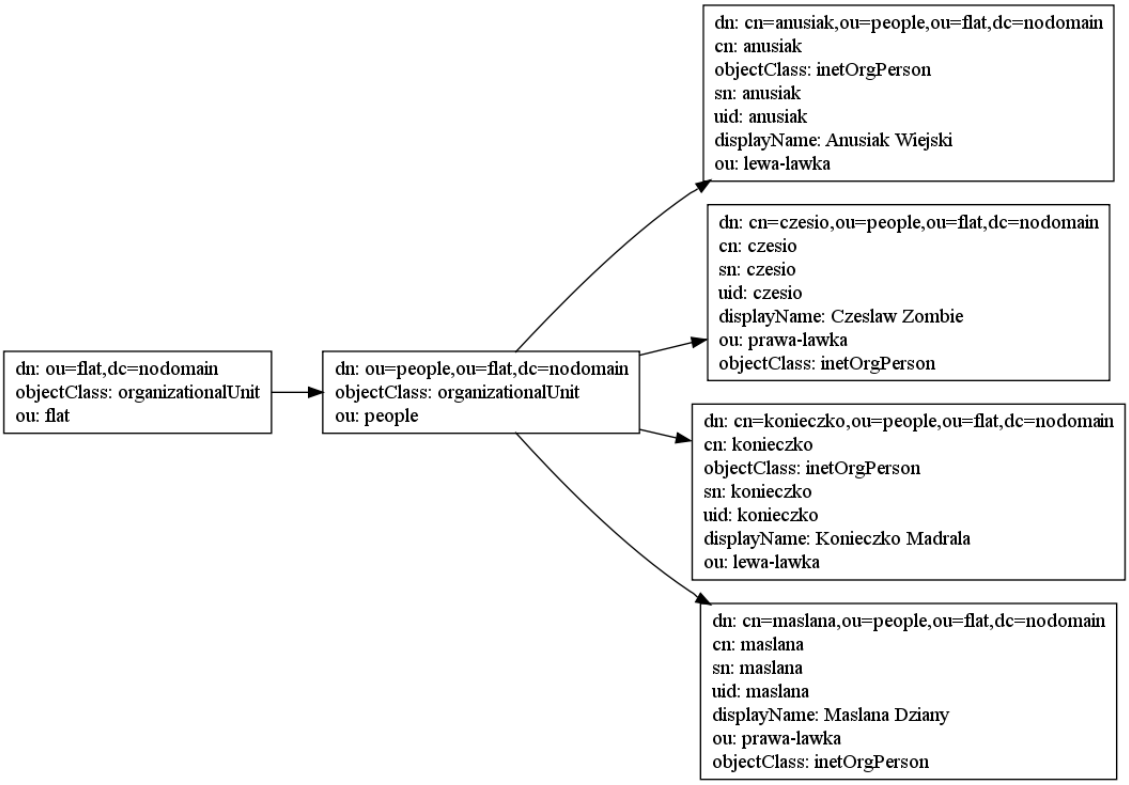

Figure 3.1: Flat DIT graph

- information duplication – the group name is repeated in every member object
- difficult group management – information about group members is not centralized, but distributed between member objects
- inefficiency – the list of unique group names has to be computed by iterating over all users

This however seems to be a common DIT layout, so the module keeps supporting it. You can use the following configuration...

```
{mod_shared_roster_ldap,[
  {ldap_base, "ou=flat,dc=nodomain"},
  {ldap_rfilter, "(objectClass=inetOrgPerson)"},
  {ldap_groupattr, "ou"},
  {ldap_memberattr, "cn"},
  {ldap_filter,  "(objectClass=inetOrgPerson)"},
  {ldap_userdesc, "displayName"}
]},
```

...to be provided with a roster as shown in figure 3.2 upon connecting as user `czesio`.

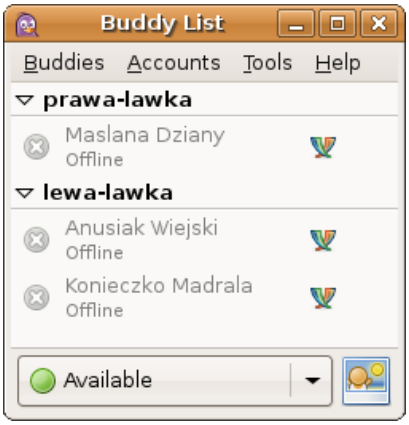

Figure 3.2: Roster from flat DIT

3.4.2 Deep DIT

This type of DIT contains distinctly typed objects for users and groups – see figure 3.3. They are shown separated into different subtrees, but it's not a requirement.

If you use the following example module configuration with it:

```
{mod_shared_roster_ldap,[
  {ldap_base, "ou=deep,dc=nodomain"},
  {ldap_rfilter, "(objectClass=groupOfUniqueNames)"},
```

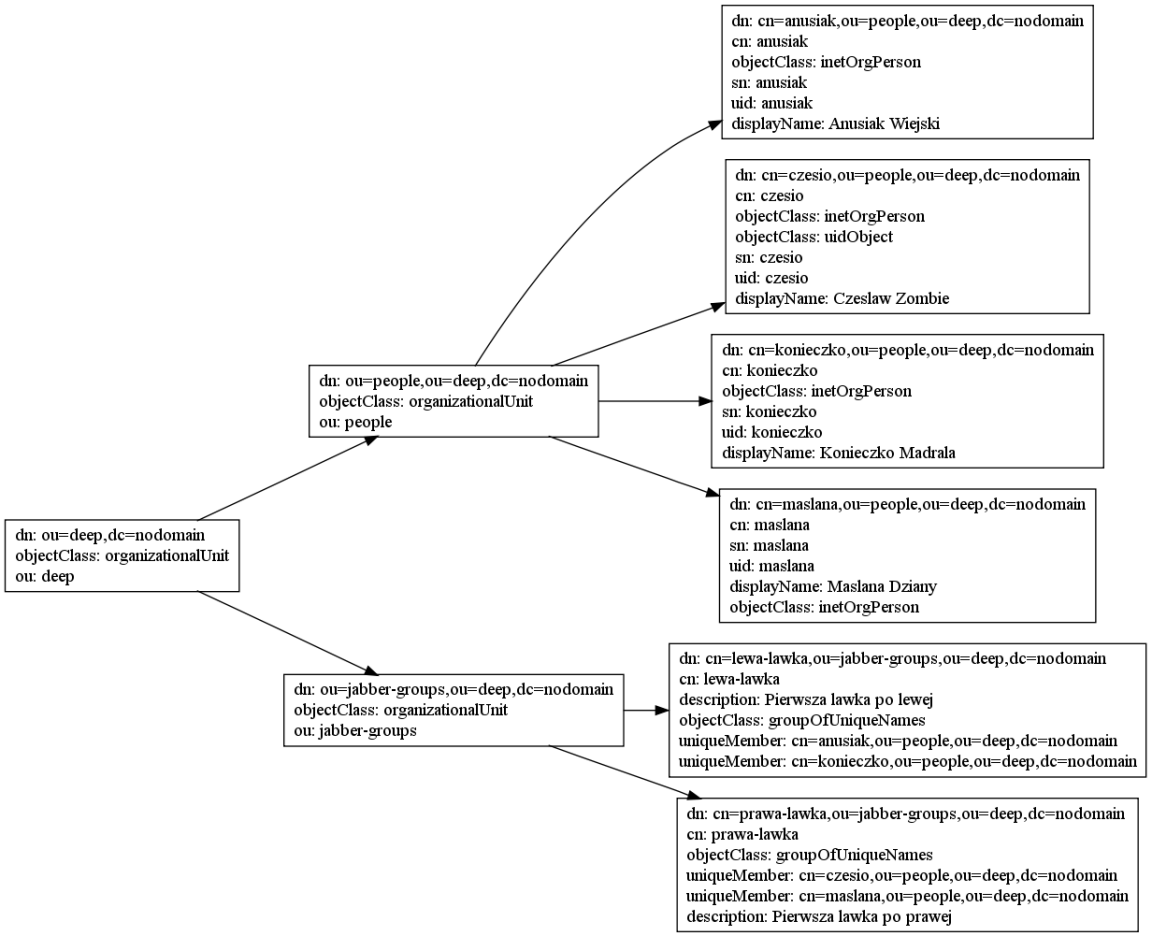

Figure 3.3: Example “deep” DIT graph

```
    {ldap_filter, ""},
    {ldap_gfilter, "(&(objectClass=groupOfUniqueNames)(cn=%g))"},
    {ldap_groupdesc, "description"},
    {ldap_memberattr, "uniqueMember"},
    {ldap_memberattr_format, "cn=%u,ou=people,ou=deep,dc=nodomain"},
    {ldap_ufilter, "(&(objectClass=inetOrgPerson)(cn=%u))"},
    {ldap_userdesc, "displayName"}
  ]},
```

. . . and connect as user `czesio`, then `ejabberd` will provide you with the roster shown in figure 3.4.

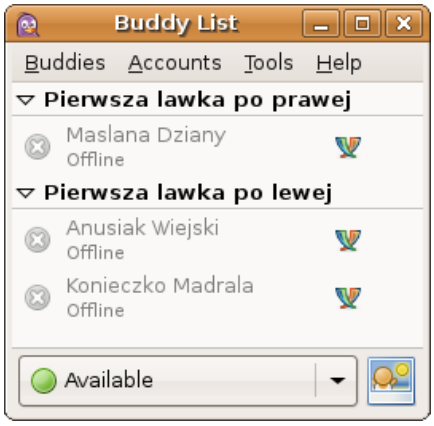

Figure 3.4: Example roster from “deep” DIT

Appendix A

Release Notes

Here are the release notes for each release, in reverse-chronological order. If you are upgrading from an older version, follow the "Upgrade instructions" for each version after the one you are upgrading from.

`0.5.3` — Changes:

- changed `group_entries_to_dict/7` to ignore (rather than crash on) UIDs which cause `jlib:nodeprep/1` to return atom `error` — for example junk entries which are over 1KiB long.
- added support for TLS connections to the LDAP server, consistent with the support in other ldap-based modules.

`0.5.2` — Changes:

- made the module compatible with ejabberd `2.1.4` (bug #312628)

`0.5.1` — Changes:

- made user `Display Names` work with mixed-case usernames. Previously they would only work for usernames which were all lower case.
- enabled stringprep driver in unit tests — this has no impact on production code.

`0.5.0` — Upgrade instructions:

- the `ldap_groupdesc` parameter now *defaults* to whatever `ldap_groupattr` is set to, rather than to `cn`. You will need to set it manually if you relied on the previous default.
- from this release on, user and group "display names" as well as group membership information is cached in memory. The list of group names is still queried on every roster retrieval.
- note that a change in this release makes the module incompatible with `ejabberd` versions below 2.0 (in case it was compatible before).

— Changes:

- changed the function which retrieves user "display names" to cache them as a dictionary, rather than a plain list, to improve lookup performance when there is a large number of users.
- changed the way group members and group "display names" are retrieved. Rather than doing it once per group, instead all members of all groups, together with group names, are retrieved in a single LDAP query and cached. See section 3.3 for more information about this.
- as a side-effect of the above change, fixed a bug where retrieving a group "display name" would only work for "Flat DIT" setups for groups with exactly one member.
- introduced a new option `ldap_group_cache_validity` which defaults to 5 minutes and lets you specify the time for which group membership and group display name information is cached.
- started using `eldap_utils:get_user_part/2` (available since `ejabberd` 2.0) rather than a local copy.
- refactored parts of documentation to reduce duplication and dispersion of related information. Also fixed a few mistakes in how filters are run.
- added upgrade instructions and missing option addition to the `0.4.0` release notes.

— Credits:

- The patch for `get_user_part/2` cleanup was contributed by Denis Kurochkin.

`0.4.0` — Upgrade instructions:

- note that there are now three required source files, not just one. See changes below, and the updated installation instructions in chapter 2.
- if you use a different "display name" than the user ID in your roster entries, then you might have to set the newly added `ldap_useruid` option to be the same as your `ldap_memberattr`. See the algorithm for retrieving the "display name" in section 3.3.

— Changes:

- added a couple of new source files: `mod_shared_roster_ldap_helpers.erl` and `mod_shared_roster_ldap.hrl`.
- added new option `ldap_memberattr_format_re` which lets you use regular expressions for extracting user IDs from attribute values, rather than simple patterns,
- added new option `ldap_auth_check` which lets you skip a verification LDAP call for each roster item,
- added new option `ldap_useruid` which lets you specify the name of the attribute which holds the ID of a user roster entry,
- introduced a new option `ldap_user_cache_validity` which defaults to 5 minutes and lets you specify the time for which user display name information is cached.
- changed the way roster item descriptions (human-friendly names) are retrieved. Instead of making an LDAP query per each roster item (which caused significant roster retrieval latency in case of large rosters) now all descriptions are retrieved with a single query for all roster items from a given domain and cached. See section 3.3 for more

information about this. This is the first and probably most significant step for fixing bug #312211[1]. Feedback is welcome on how this affects performance and memory usage.

- changed the `process_item` function (which gets called when user moves or renames items in their roster) to no longer use `Filter` for checking whether a user belongs to a group. Instead it uses the same mechanism as when loading the roster after login (described in section 3.3). I suspect this **will be slower**, sometimes significantly, than the previous approach (feedback welcome). However using this mechanism will make it easier to cache the results in the future release, leading to overall speedup.
- added instructions for installing with a binary `ejabberd` package,
- made the module log a message when it crashes,
- added unit tests for several important functions, using a couple of mocking libraries,
- simplified or eliminated some functions by extracting common code into helper functions,

— Credits:

- The patch adding support for the `ldap_memberattr_format_re` and `ldap_auth_check` options was contributed by Denis Kurochkin.

`0.3.1` — Documentation-only changes:

- added a note that defining `ldap_gfilter` is necessary when substring matching would otherwise be necessary but unavailable for `ldap_memberattr`

`0.3.0` — Changes:

- added unit tests for option parsing,
- added `ldap_ufilter` and `ldap_gfilter` options. This fixes bug #312171[2].
- changed the example a little and added another one for a flat DIT

`0.2.0` — Changes:

- applied a patch to allow the module to work with `ejabberd` 2.1.x — missing argument to `eldap:start_link`
- applied a patch from badlop and mikekaganski to nodeprep users retrieved in `get_group_users`
- optimize `get_user_roster` to only call `get_group_name` once per group. Patch provided by badlop and mikekaganski.

`0.1.1` — Initial release:

- an unversioned `mod_shared_roster_ldap.erl` imported from a webpage,
- wrote the documentation,

[1] https://alioth.debian.org/tracker/index.php?func=detail&aid=312211&group_id=100433&atid=413107
[2] https://alioth.debian.org/tracker/index.php?func=detail&aid=312171&group_id=100433&atid=413107

- there are several problems in this version:
  - does not work with `ejabberd` version `2.1` due to a missing `eldap:start_link` parameter,
  - human-readable name of each group is needlessly retrieved as many times as the group's member count,
  - roster is the same for all users — contains all groups,
  - there is just one base DN, which has to contain all user and group objects. This is very broad/inflexible and potentially inefficient.

Appendix B

Copyright Information

`mod_shared_roster_ldap` documentation.
Copyright © 2009-2010 Marcin Owsiany mailto:marcin@owsiany.pl

This document is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This document is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this document; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.