ejabberd 2.1.6

Installation and Operation Guide

ejabberd Development Team

Contents

Chapter 1

Introduction

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

`ejabberd` is cross-platform, distributed, fault-tolerant, and based on open standards to achieve real-time communication.

`ejabberd` is designed to be a rock-solid and feature rich XMPP server.

`ejabberd` is suitable for small deployments, whether they need to be scalable or not, as well as extremely big deployments.

[1] http://www.erlang.org/

1.1 Key Features

`ejabberd` is:

- Cross-platform: `ejabberd` runs under Microsoft Windows and Unix derived systems such as Linux, FreeBSD and NetBSD.
- Distributed: You can run `ejabberd` on a cluster of machines and all of them will serve the same Jabber domain(s). When you need more capacity you can simply add a new cheap node to your cluster. Accordingly, you do not need to buy an expensive high-end machine to support tens of thousands concurrent users.
- Fault-tolerant: You can deploy an `ejabberd` cluster so that all the information required for a properly working service will be replicated permanently on all nodes. This means that if one of the nodes crashes, the others will continue working without disruption. In addition, nodes also can be added or replaced 'on the fly'.
- Administrator Friendly: `ejabberd` is built on top of the Open Source Erlang. As a result you do not need to install an external database, an external web server, amongst others because everything is already included, and ready to run out of the box. Other administrator benefits include:
  - Comprehensive documentation.
  - Straightforward installers for Linux, Mac OS X, and Windows.
  - Web Administration.
  - Shared Roster Groups.
  - Command line administration tool.
  - Can integrate with existing authentication mechanisms.
  - Capability to send announce messages.
- Internationalized: `ejabberd` leads in internationalization. Hence it is very well suited in a globalized world. Related features are:
  - Translated to 25 languages.
  - Support for IDNA[2].
- Open Standards: `ejabberd` is the first Open Source Jabber server claiming to fully comply to the XMPP standard.
  - Fully XMPP compliant.
  - XML-based protocol.
  - Many protocols supported[3].

[2] http://www.ietf.org/rfc/rfc3490.txt
[3] http://www.ejabberd.im/protocols

1.2 Additional Features

Moreover, `ejabberd` comes with a wide range of other state-of-the-art features:

- Modular
  - Load only the modules you want.
  - Extend `ejabberd` with your own custom modules.
- Security
  - SASL and STARTTLS for c2s and s2s connections.
  - STARTTLS and Dialback s2s connections.
  - Web Admin accessible via HTTPS secure access.
- Databases
  - Internal database for fast deployment (Mnesia).
  - Native MySQL support.
  - Native PostgreSQL support.
  - ODBC data storage support.
  - Microsoft SQL Server support.
- Authentication
  - Internal Authentication.
  - PAM, LDAP and ODBC.
  - External Authentication script.
- Others
  - Support for virtual hosting.
  - Compressing XML streams with Stream Compression (XEP-0138[4]).
  - Statistics via Statistics Gathering (XEP-0039[5]).
  - IPv6 support both for c2s and s2s connections.
  - Multi-User Chat[6] module with support for clustering and HTML logging.
  - Users Directory based on users vCards.
  - Publish-Subscribe[7] component with support for Personal Eventing via Pubsub[8].
  - Support for web clients: HTTP Polling[9] and HTTP Binding (BOSH)[10] services.
  - IRC transport.
  - Component support: interface with networks such as AIM, ICQ and MSN installing special tranports.

[4] http://xmpp.org/extensions/xep-0138.html
[5] http://xmpp.org/extensions/xep-0039.html
[6] http://xmpp.org/extensions/xep-0045.html
[7] http://xmpp.org/extensions/xep-0060.html
[8] http://xmpp.org/extensions/xep-0163.html
[9] http://xmpp.org/extensions/xep-0025.html
[10] http://xmpp.org/extensions/xep-0206.html

Chapter 2

Installing ejabberd

2.1 Installing ejabberd with Binary Installer

Probably the easiest way to install an ejabberd instant messaging server is using the binary installer published by ProcessOne. The binary installers of released ejabberd versions are available in the ProcessOne ejabberd downloads page: http://www.process-one.net/en/ejabberd/downloads

The installer will deploy and configure a full featured ejabberd server and does not require any extra dependencies.

In *nix systems, remember to set executable the binary installer before starting it. For example:

```
chmod +x ejabberd-2.0.0_1-linux-x86-installer.bin
./ejabberd-2.0.0_1-linux-x86-installer.bin
```

ejabberd can be started manually at any time, or automatically by the operating system at system boot time.

To start and stop ejabberd manually, use the desktop shortcuts created by the installer. If the machine doesn't have a graphical system, use the scripts 'start' and 'stop' in the 'bin' directory where ejabberd is installed.

The Windows installer also adds ejabberd as a system service, and a shortcut to a debug console for experienced administrators. If you want ejabberd to be started automatically at boot time, go to the Windows service settings and set ejabberd to be automatically started. Note that the Windows service is a feature still in development, and for example it doesn't read the file ejabberdctl.cfg.

On a *nix system, if you want ejabberd to be started as daemon at boot time, copy ejabberd.init from the 'bin' directory to something like /etc/init.d/ejabberd (depending on your distribution). Create a system user called ejabberd; it will be used by the script to start the server. Then you can call /etc/inid.d/ejabberd start as root to start the server.

If `ejabberd` doesn't start correctly in Windows, try to start it using the shortcut in desktop or start menu. If the window shows error 14001, the solution is to install: "Microsoft Visual C++ 2005 SP1 Redistributable Package". You can download it from www.microsoft.com[1]. Then uninstall `ejabberd` and install it again.

If `ejabberd` doesn't start correctly and a crash dump is generated, there was a severe problem. You can try starting `ejabberd` with the script `bin/live.bat` in Windows, or with the command `bin/ejabberdctl live` in other Operating Systems. This way you see the error message provided by Erlang and can identify what is exactly the problem.

The `ejabberdctl` administration script is included in the `bin` directory. Please refer to the section 4.1 for details about `ejabberdctl`, and configurable options to fine tune the Erlang runtime system.

2.2 Installing `ejabberd` with Operating System Specific Packages

Some Operating Systems provide a specific `ejabberd` package adapted to the system architecture and libraries. It usually also checks dependencies and performs basic configuration tasks like creating the initial administrator account. Some examples are Debian and Gentoo. Consult the resources provided by your Operating System for more information.

Usually those packages create a script like `/etc/init.d/ejabberd` to start and stop `ejabberd` as a service at boot time.

2.3 Installing `ejabberd` with CEAN

CEAN[2] (Comprehensive Erlang Archive Network) is a repository that hosts binary packages from many Erlang programs, including `ejabberd` and all its dependencies. The binaries are available for many different system architectures, so this is an alternative to the binary installer and Operating System's `ejabberd` packages.

You will have to create your own `ejabberd` start script depending of how you handle your CEAN installation. The default `ejabberdctl` script is located into `ejabberd`'s priv directory and can be used as an example.

2.4 Installing `ejabberd` from Source Code

The canonical form for distribution of `ejabberd` stable releases is the source code package. Compiling `ejabberd` from source code is quite easy in *nix systems, as long as your system have all the dependencies.

[1] http://www.microsoft.com/
[2] http://cean.process-one.net/

2.4.1 Requirements

To compile **ejabberd** on a 'Unix-like' operating system, you need:

- GNU Make
- GCC
- Libexpat 1.95 or higher
- Erlang/OTP R10B-9 or higher. The recommended versions are R12B-5 and R13B04. Don't use R14A or R14B because they have a bug[3].
- OpenSSL 0.9.8 or higher, for STARTTLS, SASL and SSL encryption.
- Zlib 1.2.3 or higher, for Stream Compression support (XEP-0138[4]). Optional.
- Erlang mysql library. Optional. For MySQL authentication or storage. See section 3.2.1.
- Erlang pgsql library. Optional. For PostgreSQL authentication or storage. See section 3.2.3.
- PAM library. Optional. For Pluggable Authentication Modules (PAM). See section 3.1.4.
- GNU Iconv 1.8 or higher, for the IRC Transport (mod_irc). Optional. Not needed on systems with GNU Libc. See section 3.3.8.
- ImageMagick's Convert program. Optional. For CAPTCHA challenges. See section 3.1.8.
- exmpp 0.9.2 or higher. Optional. For import/export user data with XEP-0227[5] XML files.

2.4.2 Download Source Code

Released versions of **ejabberd** are available in the ProcessOne **ejabberd** downloads page: http://www.process-one.net/

Alternatively, the latest development source code can be retrieved from the Git repository using the commands:

```
git clone git://git.process-one.net/ejabberd/mainline.git ejabberd
cd ejabberd
git checkout -b 2.1.x origin/2.1.x
```

[3] http://www.erlang.org/cgi-bin/ezmlm-cgi/4/54598
[4] http://xmpp.org/extensions/xep-0138.html
[5] http://xmpp.org/extensions/xep-0227.html

2.4.3 Compile

To compile `ejabberd` execute the commands:

```
./configure
make
```

The build configuration script allows several options. To get the full list run the command:

```
./configure --help
```

Some options that you may be interested in modifying:

`--prefix=/` Specify the path prefix where the files will be copied when running the `make install` command.

`--enable-user[=USER]` Allow this normal system user to execute the ejabberdctl script (see section 4.1), read the configuration files, read and write in the spool directory, read and write in the log directory. The account user and group must exist in the machine before running `make install`. This account doesn't need an explicit HOME directory, because `/var/lib/ejabberd/` will be used by default.

`--enable-pam` Enable the PAM authentication method (see section 3.1.4).

`--enable-odbc or --enable-mssql` Required if you want to use an external database. See section 3.2 for more information.

`--enable-full-xml` Enable the use of XML based optimisations. It will for example use CDATA to escape characters in the XMPP stream. Use this option only if you are sure your XMPP clients include a fully compliant XML parser.

`--disable-transient-supervisors` Disable the use of Erlang/OTP supervision for transient processes.

`--enable-nif` Replaces some critical Erlang functions with equivalents written in C to improve performance. This feature requires Erlang/OTP R13B04 or higher.

2.4.4 Install

To install `ejabberd` in the destination directories, run the command:

```
make install
```

Note that you probably need administrative privileges in the system to install `ejabberd`.

The files and directories created are, by default:

`/etc/ejabberd/` Configuration directory:

- `ejabberd.cfg` ejabberd configuration file
- `ejabberdctl.cfg` Configuration file of the administration script
- `inetrc` Network DNS configuration file

`/lib/ejabberd/ ebin/` Erlang binary files (*.beam)

- `include/` Erlang header files (*.hrl)
- `priv/` Additional files required at runtime
  - `bin/` Executable programs
  - `lib/` Binary system libraries (*.so)
  - `msgs/` Translation files (*.msgs)

`/sbin/ejabberdctl` Administration script (see section 4.1)

`/share/doc/ejabberd/` Documentation of ejabberd

`/var/lib/ejabberd/` Spool directory:

- `.erlang.cookie` Erlang cookie file (see section 5.3)
- `acl.DCD, ...` Mnesia database spool files (*.DCD, *.DCL, *.DAT)

`/var/log/ejabberd/` Log directory (see section 7.1):

- `ejabberd.log` ejabberd service log
- `erlang.log` Erlang/OTP system log

2.4.5 Start

You can use the `ejabberdctl` command line administration script to start and stop `ejabberd`. If you provided the configure option `--enable-user=USER` (see 2.4.3), you can execute `ejabberdctl` with either that system account or root.

Usage example:

```
ejabberdctl start

ejabberdctl status
The node ejabberd@localhost is started with status: started
ejabberd is running in that node

ejabberdctl stop
```

If `ejabberd` doesn't start correctly and a crash dump is generated, there was a severe problem. You can try starting `ejabberd` with the command `ejabberdctl live` to see the error message provided by Erlang and can identify what is exactly the problem.

Please refer to the section 4.1 for details about `ejabberdctl`, and configurable options to fine tune the Erlang runtime system.

If you want ejabberd to be started as daemon at boot time, copy `ejabberd.init` to something like `/etc/init.d/ejabberd` (depending on your distribution). Create a system user called `ejabberd`; it will be used by the script to start the server. Then you can call `/etc/inid.d/ejabberd start` as root to start the server.

2.4.6 Specific Notes for BSD

The command to compile `ejabberd` in BSD systems is:

```
gmake
```

2.4.7 Specific Notes for Sun Solaris

You need to have `GNU install`, but it isn't included in Solaris. It can be easily installed if your Solaris system is set up for blastwave.org[6] package repository. Make sure `/opt/csw/bin` is in your `PATH` and run:

```
pkg-get -i fileutils
```

If that program is called `ginstall`, modify the `ejabberd Makefile` script to suit your system, for example:

```
cat Makefile | sed s/install/ginstall/ > Makefile.gi
```

And finally install `ejabberd` with:

```
gmake -f Makefile.gi ginstall
```

2.4.8 Specific Notes for Microsoft Windows

Requirements

To compile `ejabberd` on a Microsoft Windows system, you need:

- MS Visual C++ 6.0 Compiler
- Erlang/OTP R11B-5[7]

[6] http://www.blastwave.org/
[7] http://www.erlang.org/download.html

- Expat 2.0.0 or higher[8]
- GNU Iconv 1.9.2[9] (optional)
- Shining Light OpenSSL 0.9.8d or higher[10] (to enable SSL connections)
- Zlib 1.2.3 or higher[11]

Compilation

We assume that we will try to put as much library as possible into `C:\sdk\` to make it easier to track what is install for `ejabberd`.

1. Install Erlang emulator (for example, into `C:\sdk\erl5.5.5`).
2. Install Expat library into `C:\sdk\Expat-2.0.0` directory.

   Copy file `C:\sdk\Expat-2.0.0\Libs\libexpat.dll` to your Windows system directory (for example, `C:\WINNT` or `C:\WINNT\System32`)
3. Build and install the Iconv library into the directory `C:\sdk\GnuWin32`.

   Copy file `C:\sdk\GnuWin32\bin\lib*.dll` to your Windows system directory (more installation instructions can be found in the file README.woe32 in the iconv distribution).

   Note: instead of copying libexpat.dll and iconv.dll to the Windows directory, you can add the directories `C:\sdk\Expat-2.0.0\Libs` and `C:\sdk\GnuWin32\bin` to the `PATH` environment variable.
4. Install OpenSSL in `C:\sdk\OpenSSL` and add `C:\sdk\OpenSSL\lib\VC` to your path or copy the binaries to your system directory.
5. Install ZLib in `C:\sdk\gnuWin32`. Copy `C:\sdk\GnuWin32\bin\zlib1.dll` to your system directory. If you change your path it should already be set after libiconv install.
6. Make sure the you can access Erlang binaries from your path. For example: `set PATH=%PATH%;"C:\sdk\erl5.6.5\b`
7. Depending on how you end up actually installing the library you might need to check and tweak the paths in the file configure.erl.
8. While in the directory `ejabberd\src` run:

   ```
   configure.bat
   nmake -f Makefile.win32
   ```

9. Edit the file `ejabberd\src\ejabberd.cfg` and run

   ```
   werl -s ejabberd -name ejabberd
   ```

[8] http://sourceforge.net/project/showfiles.php?group_id=10127&package_id=11277
[9] http://www.gnu.org/software/libiconv/
[10] http://www.slproweb.com/products/Win32OpenSSL.html
[11] http://www.zlib.net/

2.5 Create a XMPP Account for Administration

You need a XMPP account and grant him administrative privileges to enter the ejabberd Web Admin:

1. Register a XMPP account on your ejabberd server, for example admin1@example.org. There are two ways to register a XMPP account:
   (a) Using ejabberdctl (see section 4.1):
   ```
   ejabberdctl register admin1 example.org FgT5bk3
   ```
   (b) Using a XMPP client and In-Band Registration (see section 3.3.18).
2. Edit the ejabberd configuration file to give administration rights to the XMPP account you created:
   ```
   {acl, admins, {user, "admin1", "example.org"}}.
   {access, configure, [{allow, admins}]}.
   ```
   You can grant administrative privileges to many XMPP accounts, and also to accounts in other XMPP servers.
3. Restart ejabberd to load the new configuration.
4. Open the Web Admin (http://server:port/admin/) in your favourite browser. Make sure to enter the *full* JID as username (in this example: admin1@example.org. The reason that you also need to enter the suffix, is because ejabberd's virtual hosting support.

2.6 Upgrading ejabberd

To upgrade an ejabberd installation to a new version, simply uninstall the old version, and then install the new one. Of course, it is important that the configuration file and Mnesia database spool directory are not removed.

ejabberd automatically updates the Mnesia table definitions at startup when needed. If you also use an external database for storage of some modules, check if the release notes of the new ejabberd version indicates you need to also update those tables.

Chapter 3

Configuring ejabberd

3.1 Basic Configuration

The configuration file will be loaded the first time you start ejabberd. The content from this file will be parsed and stored in the internal ejabberd database. Subsequently the configuration will be loaded from the database and any commands in the configuration file are appended to the entries in the database.

Note that ejabberd never edits the configuration file. So, the configuration changes done using the Web Admin are stored in the database, but are not reflected in the configuration file. If you want those changes to be use after ejabberd restart, you can either edit the configuration file, or remove all its content.

The configuration file contains a sequence of Erlang terms. Lines beginning with a '%' sign are ignored. Each term is a tuple of which the first element is the name of an option, and any further elements are that option's values. If the configuration file do not contain for instance the 'hosts' option, the old host name(s) stored in the database will be used.

You can override the old values stored in the database by adding next lines to the beginning of the configuration file:

```
override_global.
override_local.
override_acls.
```

With these lines the old global options (shared between all ejabberd nodes in a cluster), local options (which are specific for this particular ejabberd node) and ACLs will be removed before new ones are added.

3.1.1 Host Names

The option hosts defines a list containing one or more domains that ejabberd will serve.

The syntax is:

```
{hosts, [HostName, ...]}.
```

Examples:

- Serving one domain:

  ```
  {hosts, ["example.org"]}.
  ```

- Serving three domains:

  ```
  {hosts, ["example.net", "example.com", "jabber.somesite.org"]}.
  ```

3.1.2 Virtual Hosting

Options can be defined separately for every virtual host using the `host_config` option.

The syntax is:

```
{host_config, HostName, [Option, ...]}
```

Examples:

- Domain `example.net` is using the internal authentication method while domain `example.com` is using the LDAP server running on the domain `localhost` to perform authentication:

  ```
  {host_config, "example.net", [{auth_method,   internal}]}.

  {host_config, "example.com", [{auth_method,   ldap},
                                {ldap_servers,  ["localhost"]},
                                {ldap_uids,     [{"uid"}]},
                                {ldap_rootdn,   "dc=localdomain"},
                                {ldap_rootdn,   "dc=example,dc=com"},
                                {ldap_password, ""}]}.
  ```

- Domain `example.net` is using ODBC to perform authentication while domain `example.com` is using the LDAP servers running on the domains `localhost` and `otherhost`:

  ```
  {host_config, "example.net", [{auth_method, odbc},
                                {odbc_server, "DSN=ejabberd;UID=ejabberd;PWD=ejabberd"}]}.

  {host_config, "example.com", [{auth_method,   ldap},
                                {ldap_servers,  ["localhost", "otherhost"]},
                                {ldap_uids,     [{"uid"}]},
                                {ldap_rootdn,   "dc=localdomain"},
                                {ldap_rootdn,   "dc=example,dc=com"},
                                {ldap_password, ""}]}.
  ```

To define specific ejabberd modules in a virtual host, you can define the global `modules` option with the common modules, and later add specific modules to certain virtual hosts. To accomplish that, instead of defining each option in `host_config` with the general syntax

```
{OptionName, OptionValue}
```

use this syntax:

```
{{add, OptionName}, OptionValue}
```

In this example three virtual hosts have some similar modules, but there are also other different modules for some specific virtual hosts:

```
%% This ejabberd server has three vhosts:
{hosts, ["one.example.org", "two.example.org", "three.example.org"]}.

%% Configuration of modules that are common to all vhosts
{modules,
 [
  {mod_roster,     []},
  {mod_configure,  []},
  {mod_disco,      []},
  {mod_private,    []},
  {mod_time,       []},
  {mod_last,       []},
  {mod_version,    []}
 ]}.

%% Add some modules to vhost one:
{host_config, "one.example.org",
 [{{add, modules}, [
                    {mod_echo,       [{host, "echo-service.one.example.org"}]}
                    {mod_http_bind,  []},
                    {mod_logxml,     []}
                   ]
  }
 ]}.

%% Add a module just to vhost two:
{host_config, "two.example.org",
 [{{add, modules}, [
                    {mod_echo,       [{host, "mirror.two.example.org"}]}
                   ]
  }
 ]}.
```

3.1.3 Listening Ports

The option `listen` defines for which ports, addresses and network protocols `ejabberd` will listen and what services will be run on them. Each element of the list is a tuple with the following elements:

- Port number. Optionally also the IP address and/or a transport protocol.
- Listening module that serves this port.
- Options for the TCP socket and for the listening module.

The option syntax is:

```
{listen, [Listener, ...]}.
```

To define a listener there are several syntax.

```
{PortNumber, Module, [Option, ...]}
```

```
{{PortNumber, IPaddress}, Module, [Option, ...]}
```

```
{{PortNumber, TransportProtocol}, Module, [Option, ...]}
```

```
{{PortNumber, IPaddress, TransportProtocol}, Module, [Option, ...]}
```

Port Number, IP Address and Transport Protocol

The port number defines which port to listen for incoming connections. It can be a Jabber/XMPP standard port (see section 5.1) or any other valid port number.

The IP address can be represented with a string or an Erlang tuple with decimal or hexadecimal numbers. The socket will listen only in that network interface. It is possible to specify a generic address, so `ejabberd` will listen in all addresses. Depending in the type of the IP address, IPv4 or IPv6 will be used. When not specified the IP address, it will listen on all IPv4 network addresses.

Some example values for IP address:

- `"0.0.0.0"` to listen in all IPv4 network interfaces. This is the default value when no IP is specified.
- `"::"` to listen in all IPv6 network interfaces
- `"10.11.12.13"` is the IPv4 address `10.11.12.13`

- `"::FFFF:127.0.0.1"` is the IPv6 address `::FFFF:127.0.0.1/128`
- `{10, 11, 12, 13}` is the IPv4 address `10.11.12.13`
- `{0, 0, 0, 0, 0, 65535, 32512, 1}` is the IPv6 address `::FFFF:127.0.0.1/128`
- `{16#fdca, 16#8ab6, 16#a243, 16#75ef, 0, 0, 0, 1}` is the IPv6 address `FDCA:8AB6:A243:75EF::1/128`

The transport protocol can be `tcp` or `udp`. Default is `tcp`.

Listening Module

The available modules, their purpose and the options allowed by each one are:

`ejabberd_c2s` Handles c2s connections.
Options: `access`, `certfile`, `max_fsm_queue`, `max_stanza_size`, `shaper`, `starttls`, `starttls_required`, `tls`, `zlib`

`ejabberd_s2s_in` Handles incoming s2s connections.
Options: `max_stanza_size`, `shaper`

`ejabberd_service` Interacts with an external component[1] (as defined in the Jabber Component Protocol (XEP-0114[2]).
Options: `access`, `hosts`, `max_fsm_queue`, `service_check_from`, `shaper`

`ejabberd_stun` Handles STUN Binding requests as defined in RFC 5389[3].
Options: `certfile`

`ejabberd_http` Handles incoming HTTP connections.
Options: `captcha`, `certfile`, `http_bind`, `http_poll`, `request_handlers`, `tls`, `trusted_proxies`, `web_admin`

Options

This is a detailed description of each option allowed by the listening modules:

`{access, AccessName}` This option defines access to the port. The default value is `all`.

`{backlog, Value}` The backlog value defines the maximum length that the queue of pending connections may grow to. This should be increased if the server is going to handle lots of new incoming connections as they may be dropped if there is no space in the queue (and ejabberd was not able to accept them immediately). Default value is 5.

`captcha` Simple web page that allows a user to fill a CAPTCHA challenge (see section 3.1.8).

[1] http://www.ejabberd.im/tutorials-transports
[2] http://xmpp.org/extensions/xep-0114.html
[3] http://tools.ietf.org/html/rfc5389

`{certfile, Path}` Full path to a file containing the default SSL certificate. To define a certificate file specific for a given domain, use the global option `domain_certfile`.

`{hosts, [Hostname, ...], [HostOption, ...]}` The external Jabber component that connects to this `ejabberd_service` can serve one or more hostnames. As `HostOption` you can define options for the component; currently the only allowed option is the password required to the component when attempt to connect to ejabberd: `{password, Secret}`. Note that you cannot define in a single `ejabberd_service` components of different services: add an `ejabberd_service` for each service, as seen in an example below.

`http_bind` This option enables HTTP Binding (XEP-0124[4] and XEP-0206[5]) support. HTTP Bind enables access via HTTP requests to `ejabberd` from behind firewalls which do not allow outgoing sockets on port 5222.

Remember that you must also install and enable the module mod_http_bind.

If HTTP Bind is enabled, it will be available at `http://server:port/http-bind/`. Be aware that support for HTTP Bind is also needed in the XMPP client. Remark also that HTTP Bind can be interesting to host a web-based XMPP client such as JWChat[6] (check the tutorials to install JWChat with ejabberd and an embedded local web server[7] or Apache[8]).

`http_poll` This option enables HTTP Polling (XEP-0025[9]) support. HTTP Polling enables access via HTTP requests to `ejabberd` from behind firewalls which do not allow outgoing sockets on port 5222.

If HTTP Polling is enabled, it will be available at `http://server:port/http-poll/`. Be aware that support for HTTP Polling is also needed in the XMPP client. Remark also that HTTP Polling can be interesting to host a web-based XMPP client such as JWChat[10].

The maximum period of time to keep a client session active without an incoming POST request can be configured with the global option `http_poll_timeout`. The default value is five minutes. The option can be defined in `ejabberd.cfg`, expressing the time in seconds: `{http_poll_timeout, 300}.`

`{max_fsm_queue, Size}` This option specifies the maximum number of elements in the queue of the FSM (Finite State Machine). Roughly speaking, each message in such queues represents one XML stanza queued to be sent into its relevant outgoing stream. If queue size reaches the limit (because, for example, the receiver of stanzas is too slow), the FSM and the corresponding connection (if any) will be terminated and error message will be logged. The reasonable value for this option depends on your hardware configuration. However, there is no much sense to set the size above 1000 elements. This option can be specified for `ejabberd_service` and `ejabberd_c2s` listeners, or also globally for `ejabberd_s2s_out`. If the option is not specified for `ejabberd_service` or `ejabberd_c2s` listeners, the globally configured value is used. The allowed values are integers and 'undefined'. Default value: 'undefined'.

[4] http://xmpp.org/extensions/xep-0124.html
[5] http://xmpp.org/extensions/xep-0206.html
[6] http://jwchat.sourceforge.net/
[7] http://www.ejabberd.im/jwchat-localserver
[8] http://www.ejabberd.im/jwchat-apache
[9] http://xmpp.org/extensions/xep-0025.html
[10] http://jwchat.sourceforge.net/

`{max_stanza_size, Size}` This option specifies an approximate maximum size in bytes of XML stanzas. Approximate, because it is calculated with the precision of one block of read data. For example `{max_stanza_size, 65536}`. The default value is `infinity`. Recommended values are 65536 for c2s connections and 131072 for s2s connections. s2s max stanza size must always much higher than c2s limit. Change this value with extreme care as it can cause unwanted disconnect if set too low.

`{request_handlers, [ {Path, Module}, ...]}` To define one or several handlers that will serve HTTP requests. The Path is a list of strings; so the URIs that start with that Path will be served by Module. For example, if you want `mod_foo` to serve the URIs that start with `/a/b/`, and you also want `mod_http_bind` to serve the URIs `/http-bind/`, use this option: `{request_handlers, [{["a", "b"], mod_foo}, {["http-bind"], mod_http_bind}]}`

`{service_check_from, true|false}` This option can be used with `ejabberd_service` only. XEP-0114[11] requires that the domain must match the hostname of the component. If this option is set to `false`, `ejabberd` will allow the component to send stanzas with any arbitrary domain in the 'from' attribute. Only use this option if you are completely sure about it. The default value is `true`, to be compliant with XEP-0114[12].

`{shaper, none|ShaperName}` This option defines a shaper for the port (see section 3.1.6). The default value is `none`.

`starttls` This option specifies that STARTTLS encryption is available on connections to the port. You should also set the `certfile` option. You can define a certificate file for a specific domain using the global option `domain_certfile`.

`starttls_required` This option specifies that STARTTLS encryption is required on connections to the port. No unencrypted connections will be allowed. You should also set the `certfile` option. You can define a certificate file for a specific domain using the global option `domain_certfile`.

`tls` This option specifies that traffic on the port will be encrypted using SSL immediately after connecting. This was the traditional encryption method in the early Jabber software, commonly on port 5223 for client-to-server communications. But this method is nowadays deprecated and not recommended. The preferable encryption method is STARTTLS on port 5222, as defined RFC 3920: XMPP Core[13], which can be enabled in `ejabberd` with the option `starttls`. If this option is set, you should also set the `certfile` option. The option `tls` can also be used in `ejabberd_http` to support HTTPS.

`{trusted_proxies, all | [IpString]}` Specify what proxies are trusted when an HTTP request contains the header `X-Forwarded-For` You can specify `all` to allow all proxies, or specify a list of IPs in string format. The default value is: `["127.0.0.1"]`

`web_admin` This option enables the Web Admin for `ejabberd` administration which is available at `http://server:port/admin/`. Login and password are the username and password of one of the registered users who are granted access by the 'configure' access rule.

`zlib` This option specifies that Zlib stream compression (as defined in XEP-0138[14]) is available on connections to the port.

[11] http://xmpp.org/extensions/xep-0114.html
[12] http://xmpp.org/extensions/xep-0114.html
[13] http://xmpp.org/rfcs/rfc3920.html#tls
[14] http://xmpp.org/extensions/xep-0138.html

There are some additional global options that can be specified in the ejabberd configuration file (outside `listen`):

`{s2s_use_starttls, false|optional|required|required_trusted}` This option defines if s2s connections don't use STARTTLS encryption; if STARTTLS can be used optionally; if STARTTLS is required to establish the connection; or if STARTTLS is required and the remote certificate must be valid and trusted. The default value is to not use STARTTLS: `false`.

`{s2s_certfile, Path}` Full path to a file containing a SSL certificate.

`{domain_certfile, Domain, Path}` Full path to the file containing the SSL certificate for a specific domain.

`{outgoing_s2s_options, [Family, ...], Timeout}` Specify which address families to try, in what order, and connect timeout in milliseconds. By default it first tries connecting with IPv4, if that fails it tries using IPv6, with a timeout of 10000 milliseconds.

`{s2s_dns_options, [ {Property, Value}, ...]}` Define properties to use for DNS resolving. Allowed Properties are: `timeout` in seconds which default value is `10` and `retries` which default value is `2`.

`{s2s_default_policy, allow|deny}` The default policy for incoming and outgoing s2s connections to other XMPP servers. The default value is `allow`.

`{{s2s_host, Host}, allow|deny}` Defines if incoming and outgoing s2s connections with a specific remote host are allowed or denied. This allows to restrict ejabberd to only establish s2s connections with a small list of trusted servers, or to block some specific servers.

`{s2s_max_retry_delay, Seconds}` The maximum allowed delay for retry to connect after a failed connection attempt. Specified in seconds. The default value is 300 seconds (5 minutes).

`{max_fsm_queue, Size}` This option specifies the maximum number of elements in the queue of the FSM (Finite State Machine). Roughly speaking, each message in such queues represents one XML stanza queued to be sent into its relevant outgoing stream. If queue size reaches the limit (because, for example, the receiver of stanzas is too slow), the FSM and the corresponding connection (if any) will be terminated and error message will be logged. The reasonable value for this option depends on your hardware configuration. However, there is no much sense to set the size above 1000 elements. This option can be specified for `ejabberd_service` and `ejabberd_c2s` listeners, or also globally for `ejabberd_s2s_out`. If the option is not specified for `ejabberd_service` or `ejabberd_c2s` listeners, the globally configured value is used. The allowed values are integers and 'undefined'. Default value: 'undefined'.

`{route_subdomains, local|s2s}` Defines if ejabberd must route stanzas directed to subdomains locally (compliant with RFC 3920: XMPP Core[15]), or to foreign server using S2S (compliant with RFC 3920 bis[16]).

[15] http://xmpp.org/rfcs/rfc3920.html#rules.subdomain
[16] http://tools.ietf.org/html/draft-saintandre-rfc3920bis-09#section-11.3

Examples

For example, the following simple configuration defines:

- There are three domains. The default certificate file is `server.pem`. However, the c2s and s2s connections to the domain `example.com` use the file `example_com.pem`.
- Port 5222 listens for c2s connections with STARTTLS, and also allows plain connections for old clients.
- Port 5223 listens for c2s connections with the old SSL.
- Port 5269 listens for s2s connections with STARTTLS. The socket is set for IPv6 instead of IPv4.
- Port 3478 listens for STUN requests over UDP.
- Port 5280 listens for HTTP requests, and serves the HTTP Poll service.
- Port 5281 listens for HTTP requests, and serves the Web Admin using HTTPS as explained in section 4.3. The socket only listens connections to the IP address 127.0.0.1.

```
{hosts, ["example.com", "example.org", "example.net"]}.
{listen,
 [
  {5222, ejabberd_c2s, [
                        {access, c2s},
                        {shaper, c2s_shaper},
                        starttls, {certfile, "/etc/ejabberd/server.pem"},
                        {max_stanza_size, 65536}
                       ]},
  {5223, ejabberd_c2s, [
                        {access, c2s},
                        {shaper, c2s_shaper},
                        tls, {certfile, "/etc/ejabberd/server.pem"},
                        {max_stanza_size, 65536}
                       ]},
  {{5269, "::"}, ejabberd_s2s_in, [
                                   {shaper, s2s_shaper},
                                   {max_stanza_size, 131072}
                                  ]},
  {{3478, udp}, ejabberd_stun, []},
  {5280, ejabberd_http, [
                         http_poll
                        ]},
  {{5281, "127.0.0.1"}, ejabberd_http, [
                                        web_admin,
                                        tls, {certfile, "/etc/ejabberd/server.pem"},
                                       ]}
 ]
```

```
}.
{s2s_use_starttls, optional}.
{s2s_certfile, "/etc/ejabberd/server.pem"}.
{domain_certfile, "example.com", "/etc/ejabberd/example_com.pem"}.
{outgoing_s2s_options, [ipv4, ipv6], 10000}.
```

In this example, the following configuration defines that:

- c2s connections are listened for on port 5222 (all IPv4 addresses) and on port 5223 (SSL, IP 192.168.0.1 and fdca:8ab6:a243:75ef::1) and denied for the user called '`bad`'.
- s2s connections are listened for on port 5269 (all IPv4 addresses) with STARTTLS for secured traffic strictly required, and the certificates are verified. Incoming and outgoing connections of remote XMPP servers are denied, only two servers can connect: "jabber.example.org" and "example.com".
- Port 5280 is serving the Web Admin and the HTTP Polling service in all the IPv4 addresses. Note that it is also possible to serve them on different ports. The second example in section 4.3 shows how exactly this can be done.
- All users except for the administrators have a traffic of limit 1,000 Bytes/second
- The AIM transport[17] `aim.example.org` is connected to port 5233 on localhost IP addresses (127.0.0.1 and ::1) with password '`aimsecret`'.
- The ICQ transport JIT (`icq.example.org` and `sms.example.org`) is connected to port 5234 with password '`jitsecret`'.
- The MSN transport[18] `msn.example.org` is connected to port 5235 with password '`msnsecret`'.
- The Yahoo! transport[19] `yahoo.example.org` is connected to port 5236 with password '`yahoosecret`'.
- The Gadu-Gadu transport[20] `gg.example.org` is connected to port 5237 with password '`ggsecret`'.
- The Jabber Mail Component[21] `jmc.example.org` is connected to port 5238 with password '`jmcsecret`'.
- The service custom has enabled the special option to avoiding checking the `from` attribute in the packets send by this component. The component can send packets in behalf of any users from the server, or even on behalf of any server.

```
{acl, blocked, {user, "bad"}}.
{access, c2s, [{deny, blocked},
               {allow, all}]}.
{shaper, normal, {maxrate, 1000}}.
```

[17] http://www.ejabberd.im/pyaimt
[18] http://www.ejabberd.im/pymsnt
[19] http://www.ejabberd.im/yahoo-transport-2
[20] http://www.ejabberd.im/jabber-gg-transport
[21] http://www.ejabberd.im/jmc

```
{access, c2s_shaper, [{none, admin},
                      {normal, all}]}.
{listen,
 [{5222, ejabberd_c2s, [
                        {access, c2s},
                        {shaper, c2s_shaper}
                       ]},
  {{5223, {192, 168, 0, 1}}, ejabberd_c2s, [
                                            {access, c2s},
                                            ssl, {certfile, "/path/to/ssl.pem"}
                                           ]},
  {{5223, {16#fdca, 16#8ab6, 16#a243, 16#75ef, 0, 0, 0, 1}},
   ejabberd_c2s, [
                  {access, c2s},
                  ssl, {certfile, "/path/to/ssl.pem"}
                 ]},
  {5269, ejabberd_s2s_in, []},
  {{5280, {0, 0, 0, 0}}, ejabberd_http, [
                                         http_poll,
                                         web_admin
                                        ]},
  {{5233, {127, 0, 0, 1}}, ejabberd_service, [
                                              {hosts, ["aim.example.org"],
                                                 [{password, "aimsecret"}]}
                                             ]},
  {{5233, "::1"}, ejabberd_service, [
                                     {hosts, ["aim.example.org"],
                                        [{password, "aimsecret"}]}
                                    ]},
  {5234, ejabberd_service, [{hosts, ["icq.example.org", "sms.example.org"],
                             [{password, "jitsecret"}]}]},
  {5235, ejabberd_service, [{hosts, ["msn.example.org"],
                             [{password, "msnsecret"}]}]},
  {5236, ejabberd_service, [{hosts, ["yahoo.example.org"],
                             [{password, "yahoosecret"}]}]},
  {5237, ejabberd_service, [{hosts, ["gg.example.org"],
                             [{password, "ggsecret"}]}]},
  {5238, ejabberd_service, [{hosts, ["jmc.example.org"],
                             [{password, "jmcsecret"}]}]},
  {5239, ejabberd_service, [{hosts, ["custom.example.org"],
                             [{password, "customsecret"}]},
                            {service_check_from, false}]}
 ]
}.
{s2s_use_starttls, required_trusted}.
{s2s_certfile, "/path/to/ssl.pem"}.
{s2s_default_policy, deny}.
{{s2s_host,"jabber.example.org"}, allow}.
{{s2s_host,"example.com"}, allow}.
```

Note, that for services based in jabberd14 or WPJabber you have to make the transports log and do XDB by themselves:

```
<!--
   You have to add elogger and rlogger entries here when using ejabberd.
   In this case the transport will do the logging.
-->

<log id='logger'>
  <host/>
  <logtype/>
  <format>%d: [%t] (%h): %s</format>
  <file>/var/log/jabber/service.log</file>
</log>

<!--
   Some XMPP server implementations do not provide
   XDB services (for example, jabberd2 and ejabberd).
   xdb_file.so is loaded in to handle all XDB requests.
-->

<xdb id="xdb">
  <host/>
  <load>
    <!-- this is a lib of wpjabber or jabberd14 -->
    <xdb_file>/usr/lib/jabber/xdb_file.so</xdb_file>
    </load>
  <xdb_file xmlns="jabber:config:xdb_file">
    <spool><jabberd:cmdline flag='s'>/var/spool/jabber</jabberd:cmdline></spool>
  </xdb_file>
</xdb>
```

3.1.4 Authentication

The option `auth_method` defines the authentication methods that are used for user authentication. The syntax is:

```
{auth_method, [Method, ...]}.
```

The following authentication methods are supported by `ejabberd`:

- internal (default) — See section 3.1.4.
- external — See section 3.1.4.
- ldap — See section 3.2.5.
- odbc — See section 3.2.1, 3.2.3, 3.2.2 and 3.2.4.

- anonymous — See section 3.1.4.
- pam — See section 3.1.4.

Account creation is only supported by internal, external and odbc methods.

Internal

`ejabberd` uses its internal Mnesia database as the default authentication method. The value `internal` will enable the internal authentication method.

Examples:

- To use internal authentication on `example.org` and LDAP authentication on `example.net`:

```
{host_config, "example.org", [{auth_method, [internal]}]}.
{host_config, "example.net", [{auth_method, [ldap]}]}.
```

- To use internal authentication on all virtual hosts:

```
{auth_method, internal}.
```

External Script

In this authentication method, when `ejabberd` starts, it start a script, and calls it to perform authentication tasks.

The server administrator can write the external authentication script in any language. The details on the interface between ejabberd and the script are described in the `ejabberd Developers Guide`. There are also several example authentication scripts[22].

These are the specific options:

`{extauth_program, PathToScript}` Indicate in this option the full path to the external authentication script. The script must be executable by ejabberd.

`{extauth_instances, Integer}` Indicate how many instances of the script to run simultaneously to serve authentication in the virtual host. The default value is the minimum number: 1.

`{extauth_cache, false|CacheTimeInteger}` The value `false` disables the caching feature, this is the default. The integer `0` (zero) enables caching for statistics, but doesn't use that cached information to authenticate users. If another integer value is set, caching is enabled both for statistics and for authentication: the CacheTimeInteger indicates the number of seconds that ejabberd can reuse the authentication information since the user last disconnected, to verify again the user authentication without querying again the extauth script. Note: caching should not be enabled in a host if internal auth is also enabled. If caching is enabled, `mod_last` or `mod_last_odbc` must be enabled also in that vhost.

[22] http://www.ejabberd.im/extauth

This example sets external authentication, the extauth script, enables caching for 10 minutes, and starts three instances of the script for each virtual host defined in ejabberd:

```
{auth_method, [external]}.
{extauth_program, "/etc/ejabberd/JabberAuth.class.php"}.
{extauth_cache, 600}.
{extauth_instances, 3}.
```

SASL Anonymous and Anonymous Login

The value `anonymous` will enable the internal authentication method.

The anonymous authentication method can be configured with the following options. Remember that you can use the `host_config` option to set virtual host specific options (see section 3.1.2). Note that there also is a detailed tutorial regarding SASL Anonymous and anonymous login configuration[23].

`{allow_multiple_connections, false|true}` This option is only used when the anonymous mode is enabled. Setting it to `true` means that the same username can be taken multiple times in anonymous login mode if different resource are used to connect. This option is only useful in very special occasions. The default value is `false`.

`{anonymous_protocol, sasl_anon | login_anon | both}` `sasl_anon` means that the SASL Anonymous method will be used. `login_anon` means that the anonymous login method will be used. `both` means that SASL Anonymous and login anonymous are both enabled.

Those options are defined for each virtual host with the `host_config` parameter (see section 3.1.2).

Examples:

- To enable anonymous login on all virtual hosts:

  ```
  {auth_method, [anonymous]}.
  {anonymous_protocol, login_anon}.
  ```

- Similar as previous example, but limited to `public.example.org`:

  ```
  {host_config, "public.example.org", [{auth_method, [anonymous]},
                                       {anonymous_protocol, login_anon}]}.
  ```

- To enable anonymous login and internal authentication on a virtual host:

  ```
  {host_config, "public.example.org", [{auth_method, [internal,anonymous]},
                                       {anonymous_protocol, login_anon}]}.
  ```

[23] http://support.process-one.net/doc/display/MESSENGER/Anonymous+users+support

- To enable SASL Anonymous on a virtual host:

```
{host_config, "public.example.org", [{auth_method, [anonymous]},
                                     {anonymous_protocol, sasl_anon}]}.
```

- To enable SASL Anonymous and anonymous login on a virtual host:

```
{host_config, "public.example.org", [{auth_method, [anonymous]},
                                     {anonymous_protocol, both}]}.
```

- To enable SASL Anonymous, anonymous login, and internal authentication on a virtual host:

```
{host_config, "public.example.org", [{auth_method, [internal,anonymous]},
                                     {anonymous_protocol, both}]}.
```

PAM Authentication

`ejabberd` supports authentication via Pluggable Authentication Modules (PAM). PAM is currently supported in AIX, FreeBSD, HP-UX, Linux, Mac OS X, NetBSD and Solaris. PAM authentication is disabled by default, so you have to configure and compile `ejabberd` with PAM support enabled:

```
./configure --enable-pam && make install
```

Options:

`{pam_service, Name}` This option defines the PAM service name. Default is `"ejabberd"`. Refer to the PAM documentation of your operation system for more information.

`{pam_userinfotype, username|jid}` This option defines what type of information about the user ejabberd provides to the PAM service: only the username, or the user JID. Default is `username`.

Example:

```
{auth_method, [pam]}.
{pam_service, "ejabberd"}.
```

Though it is quite easy to set up PAM support in `ejabberd`, PAM itself introduces some security issues:

- To perform PAM authentication `ejabberd` uses external C-program called `epam`. By default, it is located in `/var/lib/ejabberd/priv/bin/` directory. You have to set it root on execution in the case when your PAM module requires root privileges (`pam_unix.so` for example). Also you have to grant access for `ejabberd` to this file and remove all other permissions from it. Execute with root privileges:

```
chown root:ejabberd /var/lib/ejabberd/priv/bin/epam
chmod 4750 /var/lib/ejabberd/priv/bin/epam
```

- Make sure you have the latest version of PAM installed on your system. Some old versions of PAM modules cause memory leaks. If you are not able to use the latest version, you can `kill(1) epam` process periodically to reduce its memory consumption: `ejabberd` will restart this process immediately.
- `epam` program tries to turn off delays on authentication failures. However, some PAM modules ignore this behavior and rely on their own configuration options. You can create a configuration file `ejabberd.pam`. This example shows how to turn off delays in `pam_unix.so` module:

  ```
  #%PAM-1.0
  auth        sufficient  pam_unix.so likeauth nullok nodelay
  account     sufficient  pam_unix.so
  ```

  That is not a ready to use configuration file: you must use it as a hint when building your own PAM configuration instead. Note that if you want to disable delays on authentication failures in the PAM configuration file, you have to restrict access to this file, so a malicious user can't use your configuration to perform brute-force attacks.
- You may want to allow login access only for certain users. `pam_listfile.so` module provides such functionality.
- If you use `pam_winbind` to authorise against a Windows Active Directory, then `/etc/nssswitch.conf` must be configured to use `winbind` as well.

3.1.5 Access Rules

ACL Definition

Access control in `ejabberd` is performed via Access Control Lists (ACLs). The declarations of ACLs in the configuration file have the following syntax:

```
{acl, ACLName, ACLValue}.
```

`ACLValue` can be one of the following:

`all` Matches all JIDs. Example:

```
{acl, all, all}.
```

`{user, Username}` Matches the user with the name `Username` at the first virtual host. Example:

```
{acl, admin, {user, "yozhik"}}.
```

`{user, Username, Server}` Matches the user with the JID `Username@Server` and any resource. Example:

```
{acl, admin, {user, "yozhik", "example.org"}}.
```

`{server, Server}` Matches any JID from server `Server`. Example:

```
{acl, exampleorg, {server, "example.org"}}.
```

`{resource, Resource}` Matches any JID with a resource `Resource`. Example:

```
{acl, mucklres, {resource, "muckl"}}.
```

`{shared_group, Groupname}` Matches any member of a Shared Roster Group with name `Groupname` in the virtual host. Example:

```
{acl, techgroupmembers, {shared_group, "techteam"}}.
```

`{shared_group, Groupname, Server}` Matches any member of a Shared Roster Group with name `Groupname` in the virtual host `Server`. Example:

```
{acl, techgroupmembers, {shared_group, "techteam", "example.org"}}.
```

`{user_regexp, Regexp}` Matches any local user with a name that matches `Regexp` on local virtual hosts. Example:

```
{acl, tests, {user_regexp, "^test[0-9]*$"}}.
```

`{user_regexp, UserRegexp, Server}` Matches any user with a name that matches `Regexp` at server `Server`. Example:

```
{acl, tests, {user_Userregexp, "^test", "example.org"}}.
```

`{server_regexp, Regexp}` Matches any JID from the server that matches `Regexp`. Example:

```
{acl, icq, {server_regexp, "^icq\\."}}.
```

`{resource_regexp, Regexp}` Matches any JID with a resource that matches `Regexp`. Example:

```
{acl, icq, {resource_regexp, "^laptop\\."}}.
```

`{node_regexp, UserRegexp, ServerRegexp}` Matches any user with a name that matches `UserRegexp` at any server that matches `ServerRegexp`. Example:

```
{acl, yohzik, {node_regexp, "^yohzik$", "^example.(com|org)$"}}.
```

`{user_glob, Glob}`

`{user_glob, Glob, Server}`

`{server_glob, Glob}`

`{resource_glob, Glob}`

`{node_glob, UserGlob, ServerGlob}` This is the same as above. However, it uses shell glob patterns instead of regexp. These patterns can have the following special characters:

- `*` matches any string including the null string.
- `?` matches any single character.
- `[...]` matches any of the enclosed characters. Character ranges are specified by a pair of characters separated by a '`-`'. If the first character after '`[`' is a '`!`', any character not enclosed is matched.

The following `ACLName` are pre-defined:

`all` Matches any JID.

`none` Matches no JID.

Access Rights

An entry allowing or denying access to different services. The syntax is:

```
{access, AccessName, [ {allow|deny, ACLName}, ...]}.
```

When a JID is checked to have access to `Accessname`, the server sequentially checks if that JID matches any of the ACLs that are named in the second elements of the tuples in the list. If it matches, the first element of the first matched tuple is returned, otherwise the value '`deny`' is returned.

If you define specific Access rights in a virtual host, remember that the globally defined Access rights have precedence over those. This means that, in case of conflict, the Access granted or denied in the global server is used and the Access of a virtual host doesn't have effect.

Example:

```
{access, configure, [{allow, admin}]}.
{access, something, [{deny, badmans},
                     {allow, all}]}.
```

The following `AccessName` are pre-defined:

`all` Always returns the value '`allow`'.

`none` Always returns the value '`deny`'.

Limiting Opened Sessions with ACL

The special access `max_user_sessions` specifies the maximum number of sessions (authenticated connections) per user. If a user tries to open more sessions by using different resources, the first opened session will be disconnected. The error `session replaced` will be sent to the disconnected session. The value for this option can be either a number, or `infinity`. The default value is `infinity`.

The syntax is:

```
{access, max_user_sessions, [ {MaxNumber, ACLName}, ...]}.
```

This example limits the number of sessions per user to 5 for all users, and to 10 for admins:

```
{access, max_user_sessions, [{10, admin}, {5, all}]}.
```

Several connections to a remote XMPP server with ACL

The special access `max_s2s_connections` specifies how many simultaneous S2S connections can be established to a specific remote XMPP server. The default value is `1`. There's also available the access `max_s2s_connections_per_node`.

The syntax is:

```
{access, max_s2s_connections, [ {MaxNumber, ACLName}, ...]}.
```

Examples:

- Allow up to 3 connections with each remote server:

  ```
  {access, max_s2s_connections, [{3, all}]}.
  ```

3.1.6 Shapers

Shapers enable you to limit connection traffic. The syntax is:

```
{shaper, ShaperName, Kind}.
```

Currently only one kind of shaper called `maxrate` is available. It has the following syntax:

```
{maxrate, Rate}
```

where `Rate` stands for the maximum allowed incoming rate in bytes per second. When a connection exceeds this limit, `ejabberd` stops reading from the socket until the average rate is again below the allowed maximum.

Examples:

- To define a shaper named '`normal`' with traffic speed limited to 1,000 bytes/second:

  ```
  {shaper, normal, {maxrate, 1000}}.
  ```

- To define a shaper named '`fast`' with traffic speed limited to 50,000 bytes/second:

  ```
  {shaper, fast, {maxrate, 50000}}.
  ```

3.1.7 Default Language

The option `language` defines the default language of server strings that can be seen by XMPP clients. If a XMPP client does not support `xml:lang`, the specified language is used.

The option syntax is:

```
{language, Language}.
```

The default value is `en`. In order to take effect there must be a translation file `Language.msg` in `ejabberd`'s `msgs` directory.

For example, to set Russian as default language:

```
{language, "ru"}.
```

Appendix A provides more details about internationalization and localization.

3.1.8 CAPTCHA

Some `ejabberd` modules can be configured to require a CAPTCHA challenge on certain actions. If the client does not support CAPTCHA Forms (XEP-0158[24]), a web link is provided so the user can fill the challenge in a web browser.

An example script is provided that generates the image using ImageMagick's Convert program.

The configurable options are:

`{captcha_cmd, Path}` Full path to a script that generates the image. The default value is an empty string: `""`

[24] http://xmpp.org/extensions/xep-0158.html

`{captcha_host, Host}` Host part of the URL sent to the user. You can include the port number. The URL sent to the user is formed by: `http://Host/captcha/` The default value is the first hostname configured.

Additionally, an `ejabberd_http` listener must be enabled with the `captcha` option. See section 3.1.3.

Example configuration:

```
{hosts, ["example.org"]}.

{captcha_cmd, "/lib/ejabberd/priv/bin/captcha.sh"}.
{captcha_host, "example.org:5280"}.

{listen,
 [
  ...
  {5280, ejabberd_http, [
                         captcha,
                         ...
                        ]
  }

]}.
```

3.1.9 STUN

`ejabberd` is able to act as a stand-alone STUN server (RFC 5389[25]). Currently only Binding usage is supported. In that role `ejabberd` helps clients with Jingle ICE (XEP-0176[26]) support to discover their external addresses and ports.

You should configure `ejabberd_stun` listening module as described in 3.1.3 section. If `certfile` option is defined, `ejabberd` multiplexes TCP and TLS over TCP connections on the same port. Obviously, `certfile` option is defined for `tcp` only. Note however that TCP or TLS over TCP support is not required for Binding usage and is reserved for TURN[27] functionality. Feel free to configure `udp` transport only.

Example configuration:

```
{listen,
 [
  ...
  {{3478, udp}, ejabberd_stun, []},
  {3478, ejabberd_stun, []},
```

[25] http://tools.ietf.org/html/rfc5389
[26] http://xmpp.org/extensions/xep-0176.html
[27] http://tools.ietf.org/html/draft-ietf-behave-turn-16

```
  {5349, ejabberd_stun, [{certfile, "/etc/ejabberd/server.pem"}]},
  ...
 ]
}.
```

You also need to configure DNS SRV records properly so clients can easily discover a STUN server serving your XMPP domain. Refer to section DNS Discovery of a Server[28] of RFC 5389[29] for details.

Example DNS SRV configuration:

```
_stun._udp   IN SRV  0 0 3478 stun.example.com.
_stun._tcp   IN SRV  0 0 3478 stun.example.com.
_stuns._tcp  IN SRV  0 0 5349 stun.example.com.
```

3.1.10 Include Additional Configuration Files

The option `include_config_file` in a configuration file instructs `ejabberd` to include other configuration files immediately.

The basic syntax is:

```
{include_config_file, Filename}.
```

It is possible to specify suboptions using the full syntax:

```
{include_config_file, Filename, [Suboption, ...]}.
```

The filename can be indicated either as an absolute path, or relative to the main `ejabberd` configuration file. It isn't possible to use wildcards. The file must exist and be readable.

The allowed suboptions are:

`{disallow, [Optionname, ...]}` Disallows the usage of those options in the included configuration file. The options that match this criteria are not accepted. The default value is an empty list: `[]`

`{allow_only, [Optionname, ...]}` Allows only the usage of those options in the included configuration file. The options that do not match this criteria are not accepted. The default value is: `all`

This is a basic example:

```
{include_config_file, "/etc/ejabberd/additional.cfg"}.
```

[28] http://tools.ietf.org/html/rfc5389#section-9
[29] http://tools.ietf.org/html/rfc5389

In this example, the included file is not allowed to contain a `listen` option. If such an option is present, the option will not be accepted. The file is in a subdirectory from where the main configuration file is.

```
{include_config_file, "./example.org/additional_not_listen.cfg", [{disallow, [listen]}]}.
```

In this example, `ejabberd.cfg` defines some ACL and Access rules, and later includes another file with additional rules:

```
{acl, admin, {user, "admin", "localhost"}}.
{access, announce, [{allow, admin}]}.
{include_config_file, "/etc/ejabberd/acl_and_access.cfg", [{allow_only, [acl, access]}]}.
```

and content of the file `acl_and_access.cfg` can be, for example:

```
{acl, admin, {user, "bob", "localhost"}}.
{acl, admin, {user, "jan", "localhost"}}.
```

3.1.11 Option Macros in Configuration File

In the `ejabberd` configuration file, it is possible to define a macro for a value and later use this macro when defining an option.

A macro is defined with this syntax:

```
{define_macro, 'MACRO', Value}.
```

The `MACRO` must be surrounded by single quotation marks, and all letters in uppercase; check the examples bellow. The `value` can be any valid arbitrary Erlang term.

The first definition of a macro is preserved, and additional definitions of the same macro are forgotten.

Macros are processed after additional configuration files have been included, so it is possible to use macros that are defined in configuration files included before the usage.

It isn't possible to use a macro in the definition of another macro.

There are two ways to use a macro:

`'MACRO'` You can put this instead of a value in an `ejabberd` option, and will be replaced with the `value` previously defined. If the macro is not defined previously, the program will crash and report an error.

`{use_macro, 'MACRO', Defaultvalue}` Use a macro even if it may not be defined. If the macro is not defined previously, the provided `defaultvalue` is used. This usage behaves as if it were defined and used this way:

```
    {define_macro, 'MACRO', Defaultvalue}.
    'MACRO'
```

This example shows the basic usage of a macro:

```
{define_macro, 'LOG_LEVEL_NUMBER', 5}.
{loglevel, 'LOG_LEVEL_NUMBER'}.
```

The resulting option interpreted by `ejabberd` is: {`loglevel, 5`}.

This example shows that values can be any arbitrary Erlang term:

```
{define_macro, 'USERBOB', {user, "bob", "localhost"}}.
{acl, admin, 'USERBOB'}.
```

The resulting option interpreted by `ejabberd` is: {`acl, admin, {user, "bob", "localhost"}`}.

This complex example:

```
{define_macro, 'NUMBER_PORT_C2S', 5222}.
{define_macro, 'PORT_S2S_IN', {5269, ejabberd_s2s_in, []}}.
{listen,
 [
  {'NUMBER_PORT_C2S', ejabberd_c2s, []},
  'PORT_S2S_IN',
  {{use_macro, 'NUMBER_PORT_HTTP', 5280}, ejabberd_http, []}
 ]
}.
```

produces this result after being interpreted:

```
{listen,
 [
  {5222, ejabberd_c2s, []},
  {5269, ejabberd_s2s_in, []},
  {5280, ejabberd_http, []}
 ]
}.
```

3.2 Database and LDAP Configuration

`ejabberd` uses its internal Mnesia database by default. However, it is possible to use a relational database or an LDAP server to store persistent, long-living data. `ejabberd` is very flexible: you can configure different authentication methods for different virtual hosts, you can configure

different authentication mechanisms for the same virtual host (fallback), you can set different storage systems for modules, and so forth.

The following databases are supported by `ejabberd`:

- Microsoft SQL Server[30]
- Mnesia[31]
- MySQL[32]
- Any ODBC compatible database[33]
- PostgreSQL[34]

The following LDAP servers are tested with `ejabberd`:

- Active Directory[35] (see section 3.2.5)
- OpenLDAP[36]
- Normally any LDAP compatible server should work; inform us about your success with a not-listed server so that we can list it here.

Important note about virtual hosting: if you define several domains in ejabberd.cfg (see section 3.1.1), you probably want that each virtual host uses a different configuration of database, authentication and storage, so that usernames do not conflict and mix between different virtual hosts. For that purpose, the options described in the next sections must be set inside a `host_config` for each vhost (see section 3.1.2). For example:

```
{host_config, "public.example.org", [
  {odbc_server, {pgsql, "localhost", "database-public-example-org", "ejabberd", "password"}},
  {auth_method, [odbc]}
]}.
```

3.2.1 MySQL

Although this section will describe `ejabberd`'s configuration when you want to use the native MySQL driver, it does not describe MySQL's installation and database creation. Check the MySQL documentation and the tutorial Using ejabberd with MySQL native driver[37] for information regarding these topics. Note that the tutorial contains information about `ejabberd`'s configuration which is duplicate to this section.

[30] http://www.microsoft.com/sql/
[31] http://www.erlang.org/doc/apps/mnesia/index.html
[32] http://www.mysql.com/
[33] http://en.wikipedia.org/wiki/Open_Database_Connectivity
[34] http://www.postgresql.org/
[35] http://www.microsoft.com/activedirectory/
[36] http://www.openldap.org/
[37] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver

Moreover, the file mysql.sql in the directory src/odbc might be interesting for you. This file contains the `ejabberd` schema for MySQL. At the end of the file you can find information to update your database schema.

Driver Compilation

You can skip this step if you installed `ejabberd` using a binary installer or if the binary packages of `ejabberd` you are using include support for MySQL.

1. First, install the Erlang MySQL library[38]. Make sure the compiled files are in your Erlang path; you can put them for example in the same directory as your `ejabberd` .beam files.
2. Then, configure and install `ejabberd` with ODBC support enabled (this is also needed for native MySQL support!). This can be done, by using next commands:

   ```
   ./configure --enable-odbc && make install
   ```

Database Connection

The actual database access is defined in the option `odbc_server`. Its value is used to define if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To use the native MySQL interface, you can pass a tuple of the following form as parameter:

```
{mysql, "Server", "Database", "Username", "Password"}
```

`mysql` is a keyword that should be kept as is. For example:

```
{odbc_server, {mysql, "localhost", "test", "root", "password"}}.
```

Optionally, it is possible to define the MySQL port to use. This option is only useful, in very rare cases, when you are not running MySQL with the default port setting. The `mysql` parameter can thus take the following form:

```
{mysql, "Server", Port, "Database", "Username", "Password"}
```

The `Port` value should be an integer, without quotes. For example:

```
{odbc_server, {mysql, "localhost", Port, "test", "root", "password"}}.
```

By default `ejabberd` opens 10 connections to the database for each virtual host. Use this option to modify the value:

[38] http://support.process-one.net/doc/display/CONTRIBS/Yxa

```
{odbc_pool_size, 10}.
```

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

If the connection to the database fails, `ejabberd` waits 30 seconds before retrying. You can modify this interval with this option:

```
{odbc_start_interval, 30}.
```

Authentication

The option value name may be misleading, as the `auth_method` name is used for access to a relational database through ODBC, as well as through the native MySQL interface. Anyway, the first configuration step is to define the odbc `auth_method`. For example:

```
{auth_method, [odbc]}.
```

Storage

MySQL also can be used to store information into from several `ejabberd` modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with relational databases like MySQL. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffixless or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.2 Microsoft SQL Server

Although this section will describe `ejabberd`'s configuration when you want to use Microsoft SQL Server, it does not describe Microsoft SQL Server's installation and database creation. Check the MySQL documentation and the tutorial Using ejabberd with MySQL native driver[39] for information regarding these topics. Note that the tutorial contains information about `ejabberd`'s configuration which is duplicate to this section.

Moreover, the file mssql.sql in the directory src/odbc might be interesting for you. This file contains the `ejabberd` schema for Microsoft SQL Server. At the end of the file you can find information to update your database schema.

[39] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver

Driver Compilation

You can skip this step if you installed ejabberd using a binary installer or if the binary packages of ejabberd you are using include support for ODBC.

If you want to use Microsoft SQL Server with ODBC, you need to configure, compile and install ejabberd with support for ODBC and Microsoft SQL Server enabled. This can be done, by using next commands:

```
./configure --enable-odbc --enable-mssql && make install
```

Database Connection

The configuration of Database Connection for a Microsoft SQL Server is the same as the configuration for ODBC compatible servers (see section 3.2.4).

Authentication

The configuration of Authentication for a Microsoft SQL Server is the same as the configuration for ODBC compatible servers (see section 3.2.4).

Storage

Microsoft SQL Server also can be used to store information into from several ejabberd modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with relational databases like Microsoft SQL Server. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.3 PostgreSQL

Although this section will describe ejabberd's configuration when you want to use the native PostgreSQL driver, it does not describe PostgreSQL's installation and database creation. Check the PostgreSQL documentation and the tutorial Using ejabberd with MySQL native driver[40] for information regarding these topics. Note that the tutorial contains information about ejabberd's configuration which is duplicate to this section.

Also the file pg.sql in the directory src/odbc might be interesting for you. This file contains the ejabberd schema for PostgreSQL. At the end of the file you can find information to update your database schema.

[40] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver

Driver Compilation

You can skip this step if you installed `ejabberd` using a binary installer or if the binary packages of `ejabberd` you are using include support for PostgreSQL.

1. First, install the Erlang pgsql library from ejabberd-modules SVN repository[41]. Make sure the compiled files are in your Erlang path; you can put them for example in the same directory as your `ejabberd` .beam files.
2. Then, configure, compile and install `ejabberd` with ODBC support enabled (this is also needed for native PostgreSQL support!). This can be done, by using next commands:

   ```
   ./configure --enable-odbc && make install
   ```

Database Connection

The actual database access is defined in the option `odbc_server`. Its value is used to define if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To use the native PostgreSQL interface, you can pass a tuple of the following form as parameter:

```
{pgsql, "Server", "Database", "Username", "Password"}
```

`pgsql` is a keyword that should be kept as is. For example:

```
{odbc_server, {pgsql, "localhost", "database", "ejabberd", "password"}}.
```

Optionally, it is possible to define the PostgreSQL port to use. This option is only useful, in very rare cases, when you are not running PostgreSQL with the default port setting. The `pgsql` parameter can thus take the following form:

```
{pgsql, "Server", Port, "Database", "Username", "Password"}
```

The `Port` value should be an integer, without quotes. For example:

```
{odbc_server, {pgsql, "localhost", 5432, "database", "ejabberd", "password"}}.
```

By default `ejabberd` opens 10 connections to the database for each virtual host. Use this option to modify the value:

```
{odbc_pool_size, 10}.
```

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

[41] http://www.ejabberd.im/ejabberd-modules/

Authentication

The option value name may be misleading, as the `auth_method` name is used for access to a relational database through ODBC, as well as through the native PostgreSQL interface. Anyway, the first configuration step is to define the odbc `auth_method`. For example:

```
{auth_method, [odbc]}.
```

Storage

PostgreSQL also can be used to store information into from several `ejabberd` modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with relational databases like PostgreSQL. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.4 ODBC Compatible

Although this section will describe `ejabberd`'s configuration when you want to use the ODBC driver, it does not describe the installation and database creation of your database. Check the documentation of your database. The tutorial Using ejabberd with MySQL native driver[42] also can help you. Note that the tutorial contains information about `ejabberd`'s configuration which is duplicate to this section.

Driver Compilation

You can skip this step if you installed `ejabberd` using a binary installer or if the binary packages of `ejabberd` you are using include support for ODBC.

1. First, install the Erlang MySQL library[43]. Make sure the compiled files are in your Erlang path; you can put them for example in the same directory as your `ejabberd` .beam files.

2. Then, configure, compile and install `ejabberd` with ODBC support enabled. This can be done, by using next commands:

   ```
   ./configure --enable-odbc && make install
   ```

[42] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver
[43] http://support.process-one.net/doc/display/CONTRIBS/Yxa

Database Connection

The actual database access is defined in the option `odbc_server`. Its value is used to defined if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To use a relational database through ODBC, you can pass the ODBC connection string as `odbc_server` parameter. For example:

```
{odbc_server, "DSN=database;UID=ejabberd;PWD=password"}.
```

By default `ejabberd` opens 10 connections to the database for each virtual host. Use this option to modify the value:

```
{odbc_pool_size, 10}.
```

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

Authentication

The first configuration step is to define the odbc `auth_method`. For example:

```
{auth_method, [odbc]}.
```

Storage

An ODBC compatible database also can be used to store information into from several `ejabberd` modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with ODBC compatible relational databases. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.5 LDAP

`ejabberd` has built-in LDAP support. You can authenticate users against LDAP server and use LDAP directory as vCard storage.

Usually `ejabberd` treats LDAP as a read-only storage: it is possible to consult data, but not possible to create accounts or edit vCard that is stored in LDAP. However, it is possible to change passwords if `mod_register` module is enabled and LDAP server supports RFC 3062[44].

[44] http://tools.ietf.org/html/rfc3062

Connection

Two connections are established to the LDAP server per vhost, one for authentication and other for regular calls.

Parameters:

`{ldap_servers, [Servers, ...]}` List of IP addresses or DNS names of your LDAP servers. This option is required.

`{ldap_encrypt, none|tls}` Type of connection encryption to the LDAP server. Allowed values are: `none`, `tls`. The value `tls` enables encryption by using LDAP over SSL. Note that STARTTLS encryption is not supported. The default value is: `none`.

`{ldap_tls_verify, false|soft|hard}` This option specifies whether to verify LDAP server certificate or not when TLS is enabled. When `hard` is enabled `ejabberd` doesn't proceed if a certificate is invalid. When `soft` is enabled `ejabberd` proceeds even if check fails. The default is `false` which means no checks are performed.

`{ldap_port, Number}` Port to connect to your LDAP server. The default port is 389 if encryption is disabled; and 636 if encryption is enabled. If you configure a value, it is stored in `ejabberd`'s database. Then, if you remove that value from the configuration file, the value previously stored in the database will be used instead of the default port.

`{ldap_rootdn, RootDN}` Bind DN. The default value is `""` which means 'anonymous connection'.

`{ldap_password, Password}` Bind password. The default value is `""`.

Example:

```
{auth_method, ldap}.
{ldap_servers, ["ldap.example.org"]}.
{ldap_port, 389}.
{ldap_rootdn, "cn=Manager,dc=domain,dc=org"}.
{ldap_password, "secret"}.
```

Authentication

You can authenticate users against an LDAP directory. Note that current LDAP implementation does not support SASL authentication.

Available options are:

`{ldap_base, Base}` LDAP base directory which stores users accounts. This option is required.

`{ldap_uids, [ {ldap_uidattr}| {ldap_uidattr, ldap_uidattr_format}, ...]}` LDAP attribute which holds a list of attributes to use as alternatives for getting the JID. The default attributes are `[{"uid", "%u"}]`. The attributes are of the form: `[{ldap_uidattr}]` or `[{ldap_uidattr, ldap_uidattr_format}]`. You can use as many comma separated attributes as needed. The values for `ldap_uidattr` and `ldap_uidattr_format` are described as follow:

`ldap_uidattr` LDAP attribute which holds the user's part of a JID. The default value is `"uid"`.

`ldap_uidattr_format` Format of the `ldap_uidattr` variable. The format *must* contain one and only one pattern variable `"%u"` which will be replaced by the user's part of a JID. For example, `"%u@example.org"`. The default value is `"%u"`.

`{ldap_filter, Filter}` RFC 4515[45] LDAP filter. The default Filter value is: `undefined`. Example: `"(&(objectClass=shadowAccount)(memberOf=Jabber Users))"`. Please, do not forget to close brackets and do not use superfluous whitespaces. Also you *must not* use `ldap_uidattr` attribute in filter because this attribute will be substituted in LDAP filter automatically.

`{ldap_dn_filter, {Filter, FilterAttrs }}` This filter is applied on the results returned by the main filter. This filter performs additional LDAP lookup to make the complete result. This is useful when you are unable to define all filter rules in `ldap_filter`. You can define `"%u"`, `"%d"`, `"%s"` and `"%D"` pattern variables in Filter: `"%u"` is replaced by a user's part of a JID, `"%d"` is replaced by the corresponding domain (virtual host), all `"%s"` variables are consecutively replaced by values of FilterAttrs attributes and `"%D"` is replaced by Distinguished Name. By default `ldap_dn_filter` is undefined. Example:

```
{ldap_dn_filter, {"(&(name=%s)(owner=%D)(user=%u@%d))", ["sn"]}}.
```

Since this filter makes additional LDAP lookups, use it only in the last resort: try to define all filter rules in `ldap_filter` if possible.

`{ldap_local_filter, Filter}` If you can't use `ldap_filter` due to performance reasons (the LDAP server has many users registered), you can use this local filter. The local filter checks an attribute in ejabberd, not in LDAP, so this limits the load on the LDAP directory. The default filter is: `undefined`. Example values:

```
{ldap_local_filter, {notequal, {"accountStatus",["disabled"]}}}.
{ldap_local_filter, {equal, {"accountStatus",["enabled"]}}}.
{ldap_local_filter, undefined}.
```

Examples

Common example Let's say `ldap.example.org` is the name of our LDAP server. We have users with their passwords in `"ou=Users,dc=example,dc=org"` directory. Also we have addressbook, which contains users emails and their additional infos in `"ou=AddressBook,dc=example,dc=org"` directory. The connection to the LDAP server is encrypted using TLS, and using the custom port 6123. Corresponding authentication section should looks like this:

[45] http://tools.ietf.org/html/rfc4515

```
%% Authentication method
{auth_method, ldap}.
%% DNS name of our LDAP server
{ldap_servers, ["ldap.example.org"]}.
%% Bind to LDAP server as "cn=Manager,dc=example,dc=org" with password "secret"
{ldap_rootdn, "cn=Manager,dc=example,dc=org"}.
{ldap_password, "secret"}.
{ldap_encrypt, tls}.
{ldap_port, 6123}.
%% Define the user's base
{ldap_base, "ou=Users,dc=example,dc=org"}.
%% We want to authorize users from 'shadowAccount' object class only
{ldap_filter, "(objectClass=shadowAccount)"}.
```

Now we want to use users LDAP-info as their vCards. We have four attributes defined in our LDAP schema: `"mail"` — email address, `"givenName"` — first name, `"sn"` — second name, `"birthDay"` — birthday. Also we want users to search each other. Let's see how we can set it up:

```
{modules,
 [
  ...
  {mod_vcard_ldap,
   [
    %% We use the same server and port, but want to bind anonymously because
    %% our LDAP server accepts anonymous requests to
    %% "ou=AddressBook,dc=example,dc=org" subtree.
    {ldap_rootdn, ""},
    {ldap_password, ""},
    %% define the addressbook's base
    {ldap_base, "ou=AddressBook,dc=example,dc=org"},
    %% uidattr: user's part of JID is located in the "mail" attribute
    %% uidattr_format: common format for our emails
    {ldap_uids, [{"mail", "%u@mail.example.org"}]},
    %% We have to define empty filter here, because entries in addressbook does not
    %% belong to shadowAccount object class
    {ldap_filter, ""},
    %% Now we want to define vCard pattern
    {ldap_vcard_map,
     [{"NICKNAME", "%u", []}, % just use user's part of JID as his nickname
      {"GIVEN", "%s", ["givenName"]},
      {"FAMILY", "%s", ["sn"]},
      {"FN", "%s, %s", ["sn", "givenName"]}, % example: "Smith, John"
      {"EMAIL", "%s", ["mail"]},
      {"BDAY", "%s", ["birthDay"]}]},
    %% Search form
    {ldap_search_fields,
     [{"User", "%u"},
```

```
      {"Name", "givenName"},
      {"Family Name", "sn"},
      {"Email", "mail"},
      {"Birthday", "birthDay"}]},
    %% vCard fields to be reported
    %% Note that JID is always returned with search results
    {ldap_search_reported,
     [{"Full Name", "FN"},
      {"Nickname", "NICKNAME"},
      {"Birthday", "BDAY"}]}
  ]},
  ...
]}.
```

Note that `mod_vcard_ldap` module checks for the existence of the user before searching in his information in LDAP.

Active Directory Active Directory is just an LDAP-server with predefined attributes. A sample configuration is shown below:

```
{auth_method, ldap}.
{ldap_servers, ["office.org"]}.    % List of LDAP servers
{ldap_base, "DC=office,DC=org"}. % Search base of LDAP directory
{ldap_rootdn, "CN=Administrator,CN=Users,DC=office,DC=org"}. % LDAP manager
{ldap_password, "*******"}. % Password to LDAP manager
{ldap_uids, [{"sAMAccountName"}]}.
{ldap_filter, "(memberOf=*)"}.

{modules,
 [
  ...
  {mod_vcard_ldap,
   [{ldap_vcard_map,
     [{"NICKNAME", "%u", []},
      {"GIVEN", "%s", ["givenName"]},
      {"MIDDLE", "%s", ["initials"]},
      {"FAMILY", "%s", ["sn"]},
      {"FN", "%s", ["displayName"]},
      {"EMAIL", "%s", ["mail"]},
      {"ORGNAME", "%s", ["company"]},
      {"ORGUNIT", "%s", ["department"]},
      {"CTRY", "%s", ["c"]},
      {"LOCALITY", "%s", ["l"]},
      {"STREET", "%s", ["streetAddress"]},
      {"REGION", "%s", ["st"]},
      {"PCODE", "%s", ["postalCode"]},
      {"TITLE", "%s", ["title"]},
```

```
      {"URL", "%s", ["wWWHomePage"]},
      {"DESC", "%s", ["description"]},
      {"TEL", "%s", ["telephoneNumber"]}]},
    {ldap_search_fields,
     [{"User", "%u"},
      {"Name", "givenName"},
      {"Family Name", "sn"},
      {"Email", "mail"},
      {"Company", "company"},
      {"Department", "department"},
      {"Role", "title"},
      {"Description", "description"},
      {"Phone", "telephoneNumber"}]},
    {ldap_search_reported,
     [{"Full Name", "FN"},
      {"Nickname", "NICKNAME"},
      {"Email", "EMAIL"}]}
  ]},
  ...
]}.
```

3.3 Modules Configuration

The option `modules` defines the list of modules that will be loaded after `ejabberd`'s startup. Each entry in the list is a tuple in which the first element is the name of a module and the second is a list of options for that module.

The syntax is:

```
{modules, [ {ModuleName, ModuleOptions}, ...]}.
```

Examples:

- In this example only the module `mod_echo` is loaded and no module options are specified between the square brackets:

  ```
  {modules,
   [
    {mod_echo,      []}
   ]}.
  ```

- In the second example the modules `mod_echo`, `mod_time`, and `mod_version` are loaded without options. Remark that, besides the last entry, all entries end with a comma:

  ```
  {modules,
   [
  ```

```
 {mod_echo,      []},
 {mod_time,      []},
 {mod_version,   []}
]}.
```

3.3.1 Modules Overview

The following table lists all modules included in `ejabberd`.

Module	Feature	Dependencies
`mod_adhoc`	Ad-Hoc Commands (XEP-0050[46])	
`mod_announce`	Manage announcements	recommends `mod_adhoc`
`mod_caps`	Entity Capabilities (XEP-0115[47])	
`mod_configure`	Server configuration using Ad-Hoc	`mod_adhoc`
`mod_disco`	Service Discovery (XEP-0030[48])	
`mod_echo`	Echoes XMPP stanzas	
`mod_http_bind`	XMPP over Bosh service (HTTP Binding)	
`mod_http_fileserver`	Small HTTP file server	
`mod_irc`	IRC transport	
`mod_last`	Last Activity (XEP-0012[49])	
`mod_last_odbc`	Last Activity (XEP-0012[50])	supported DB (*)
`mod_muc`	Multi-User Chat (XEP-0045[51])	
`mod_muc_log`	Multi-User Chat room logging	`mod_muc`
`mod_offline`	Offline message storage (XEP-0160[52])	
`mod_offline_odbc`	Offline message storage (XEP-0160[53])	supported DB (*)
`mod_ping`	XMPP Ping and periodic keepalives (XEP-0199[54])	
`mod_privacy`	Blocking Communication (XMPP IM)	
`mod_privacy_odbc`	Blocking Communication (XMPP IM)	supported DB (*)
`mod_private`	Private XML Storage (XEP-0049[55])	
`mod_private_odbc`	Private XML Storage (XEP-0049[56])	supported DB (*)
`mod_proxy65`	SOCKS5 Bytestreams (XEP-0065[57])	
`mod_pubsub`	Pub-Sub (XEP-0060[58]), PEP (XEP-0163[59])	`mod_caps`
`mod_pubsub_odbc`	Pub-Sub (XEP-0060[60]), PEP (XEP-0163[61])	supported DB (*) and `mod_c`
`mod_register`	In-Band Registration (XEP-0077[62])	
`mod_register_web`	Web for Account Registrations	
`mod_roster`	Roster management (XMPP IM)	
`mod_roster_odbc`	Roster management (XMPP IM)	supported DB (*)
`mod_service_log`	Copy user messages to logger service	
`mod_shared_roster`	Shared roster management	`mod_roster` or `mod_roster_odbc`
`mod_shared_roster_ldap`	LDAP Shared roster management	`mod_roster` or `mod_roster_odbc`
`mod_sic`	Server IP Check (XEP-0279[63])	
`mod_stats`	Statistics Gathering (XEP-0039[64])	
`mod_time`	Entity Time (XEP-0202[65])	
`mod_vcard`	vcard-temp (XEP-0054[66])	
`mod_vcard_ldap`	vcard-temp (XEP-0054[67])	LDAP server
`mod_vcard_odbc`	vcard-temp (XEP-0054[68])	supported DB (*)
`mod_vcard_xupdate`	vCard-Based Avatars (XEP-0153[69])	`mod_vcard` or `mod_vcard_odb`
`mod_version`	Software Version (XEP-0092[70])	

- (*) This module requires a supported database. For a list of supported databases, see section 3.2.

You can see which database backend each module needs by looking at the suffix:

- No suffix, this means that the modules uses Erlang's built-in database Mnesia as backend.
- '_odbc', this means that the module needs a supported database (see 3.2) as backend.
- '_ldap', this means that the module needs an LDAP server as backend.

If you want to, it is possible to use a relational database to store the tables created by some ejabberd modules. You can do this by changing the module name to a name with an `_odbc` suffix in `ejabberd` config file. You can use a relational database for the following data:

- Last connection date and time: Use `mod_last_odbc` instead of `mod_last`.
- Offline messages: Use `mod_offline_odbc` instead of `mod_offline`.
- Rosters: Use `mod_roster_odbc` instead of `mod_roster`.
- Users' VCARD: Use `mod_vcard_odbc` instead of `mod_vcard`.
- Private XML storage: Use `mod_private_odbc` instead of `mod_private`.
- User rules for blocking communications: Use `mod_privacy_odbc` instead of `mod_privacy`.
- Pub-Sub nodes, items and subscriptions: Use `mod_pubsub_odbc` instead of `mod_pubsub`.

You can find more contributed modules[71] on the `ejabberd` website. Please remember that these contributions might not work or that they can contain severe bugs and security leaks. Therefore, use them at your own risk!

3.3.2 Common Options

The following options are used by many modules. Therefore, they are described in this separate section.

`iqdisc`

Many modules define handlers for processing IQ queries of different namespaces to this server or to a user (e. g. to `example.org` or to `user@example.org`). This option defines processing discipline for these queries.

The syntax is:

```
{iqdisc, Value}
```

Possible `Value` are:

[71] http://www.ejabberd.im/contributions

`no_queue` All queries of a namespace with this processing discipline are processed immediately. This also means that no other packets can be processed until this one has been completely processed. Hence this discipline is not recommended if the processing of a query can take a relatively long time.

`one_queue` In this case a separate queue is created for the processing of IQ queries of a namespace with this discipline. In addition, the processing of this queue is done in parallel with that of other packets. This discipline is most recommended.

`{queues, N}` N separate queues are created to process the queries. The queries are thus process in parallel, but in a controlled way.

`parallel` For every packet with this discipline a separate Erlang process is spawned. Consequently, all these packets are processed in parallel. Although spawning of Erlang process has a relatively low cost, this can break the server's normal work, because the Erlang emulator has a limit on the number of processes (32000 by default).

Example:

```
{modules,
 [
  ...
  {mod_time, [{iqdisc, no_queue}]},
  ...
 ]}.
```

`host`

This option defines the Jabber ID of a service provided by an `ejabberd` module.

The syntax is:

```
{host, HostName}
```

If you include the keyword "@HOST@" in the HostName, it is replaced at start time with the real virtual host string.

This example configures the echo module to provide its echoing service in the Jabber ID `mirror.example.org`:

```
{modules,
 [
  ...
  {mod_echo, [{host, "mirror.example.org"}]},
  ...
 ]}.
```

However, if there are several virtual hosts and this module is enabled in all of them, the "@HOST@" keyword must be used:

```
{modules,
 [
  ...
  {mod_echo, [{host, "mirror.@HOST@"}]},
  ...
 ]}.
```

3.3.3 mod_announce

This module enables configured users to broadcast announcements and to set the message of the day (MOTD). Configured users can perform these actions with a XMPP client either using Ad-hoc commands or sending messages to specific JIDs.

The Ad-hoc commands are listed in the Server Discovery. For this feature to work, `mod_adhoc` must be enabled.

The specific JIDs where messages can be sent are listed bellow. The first JID in each entry will apply only to the specified virtual host `example.org`, while the JID between brackets will apply to all virtual hosts in ejabberd.

`example.org/announce/all (example.org/announce/all-hosts/all)` The message is sent to all registered users. If the user is online and connected to several resources, only the resource with the highest priority will receive the message. If the registered user is not connected, the message will be stored offline in assumption that offline storage (see section 3.3.12) is enabled.

`example.org/announce/online (example.org/announce/all-hosts/online)` The message is sent to all connected users. If the user is online and connected to several resources, all resources will receive the message.

`example.org/announce/motd (example.org/announce/all-hosts/motd)` The message is set as the message of the day (MOTD) and is sent to users when they login. In addition the message is sent to all connected users (similar to `announce/online`).

`example.org/announce/motd/update (example.org/announce/all-hosts/motd/update)` The message is set as message of the day (MOTD) and is sent to users when they login. The message is *not sent* to any currently connected user.

`example.org/announce/motd/delete (example.org/announce/all-hosts/motd/delete)` Any message sent to this JID removes the existing message of the day (MOTD).

Options:

`{access, AccessName}` This option specifies who is allowed to send announcements and to set the message of the day (by default, nobody is able to send such messages).

Examples:

- Only administrators can send announcements:

```
{access, announce, [{allow, admins}]}.

{modules,
 [
  ...
  {mod_adhoc, []},
  {mod_announce, [{access, announce}]},
  ...
 ]}.
```

- Administrators as well as the direction can send announcements:

```
{acl, direction, {user, "big_boss", "example.org"}}.
{acl, direction, {user, "assistant", "example.org"}}.
{acl, admins, {user, "admin", "example.org"}}.

{access, announce, [{allow, admins},
                    {allow, direction}]}.

{modules,
 [
  ...
  {mod_adhoc, []},
  {mod_announce, [{access, announce}]},
  ...
 ]}.
```

Note that `mod_announce` can be resource intensive on large deployments as it can broadcast lot of messages. This module should be disabled for instances of `ejabberd` with hundreds of thousands users.

3.3.4 `mod_disco`

This module adds support for Service Discovery (XEP-0030[72]). With this module enabled, services on your server can be discovered by XMPP clients. Note that `ejabberd` has no modules with support for the superseded Jabber Browsing (XEP-0011[73]) and Agent Information (XEP-0094[74]). Accordingly, XMPP clients need to have support for the newer Service Discovery protocol if you want them be able to discover the services you offer.

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for Service Discovery (`http://jabber.org/proto` and `http://jabber.org/protocol/disco#info`) IQ queries (see section 3.3.2).

[72] http://xmpp.org/extensions/xep-0030.html
[73] http://xmpp.org/extensions/xep-0011.html
[74] http://xmpp.org/extensions/xep-0094.html

`{extra_domains, [Domain, ...]}` With this option, you can specify a list of extra domains that are added to the Service Discovery item list.

`{server_info, [ {Modules, Field, [Value, ...]}, ... ]}` Specify additional information about the server, as described in Contact Addresses for XMPP Services (XEP-0157[75]). `Modules` can be the keyword 'all', in which case the information is reported in all the services; or a list of `ejabberd` modules, in which case the information is only specified for the services provided by those modules. Any arbitrary `Field` and `Value` can be specified, not only contact addresses.

Examples:

- To serve a link to the Jabber User Directory on `jabber.org`:

```
{modules,
 [
  ...
  {mod_disco, [{extra_domains, ["users.jabber.org"]}]},
  ...
 ]}.
```

- To serve a link to the transports on another server:

```
{modules,
 [
  ...
  {mod_disco, [{extra_domains, ["icq.example.com",
                                "msn.example.com"]}]},
  ...
 ]}.
```

- To serve a link to a few friendly servers:

```
{modules,
 [
  ...
  {mod_disco, [{extra_domains, ["example.org",
                                "example.com"]}]},
  ...
 ]}.
```

- With this configuration, all services show abuse addresses, feedback address on the main server, and admin addresses for both the main server and the vJUD service:

```
{modules,
 [
  ...
  {mod_disco, [{server_info, [
```

[75] http://xmpp.org/extensions/xep-0157.html

```
          {all,
           "abuse-addresses",
           ["mailto:abuse@shakespeare.lit"]},
          {[mod_muc],
           "Web chatroom logs",
           ["http://www.example.org/muc-logs"]},
          {[mod_disco],
           "feedback-addresses",
           ["http://shakespeare.lit/feedback.php", "mailto:feedback@shakespeare.lit", "xmpp:f
          {[mod_disco, mod_vcard],
           "admin-addresses",
           ["mailto:xmpp@shakespeare.lit", "xmpp:admins@shakespeare.lit"]}
       ]}]},
       ...
      ]}.
```

3.3.5 mod_echo

This module simply echoes any XMPP packet back to the sender. This mirror can be of interest for `ejabberd` and XMPP client debugging.

Options:

`{host, HostName}` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`echo.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

Example: Mirror, mirror, on the wall, who is the most beautiful of them all?

```
{modules,
 [
  ...
  {mod_echo, [{host, "mirror.example.org"}]},
  ...
 ]}.
```

3.3.6 mod_http_bind

This module implements XMPP over Bosh (formerly known as HTTP Binding) as defined in XEP-0124[76] and XEP-0206[77]. It extends ejabberd's built in HTTP service with a configurable resource at which this service will be hosted.

To use HTTP-Binding, enable the module:

[76] http://xmpp.org/extensions/xep-0124.html
[77] http://xmpp.org/extensions/xep-0206.html

```
{modules,
 [
  ...
  {mod_http_bind, []},
  ...
]}.
```

and add `http_bind` in the HTTP service. For example:

```
{listen,
 [
  ...
  {5280, ejabberd_http, [
                         http_bind,
                         http_poll,
                         web_admin
                        ]
  },
  ...
]}.
```

With this configuration, the module will serve the requests sent to `http://example.org:5280/http-bind/` Remember that this page is not designed to be used by web browsers, it is used by XMPP clients that support XMPP over Bosh.

If you want to set the service in a different URI path or use a different module, you can configure it manually using the option `request_handlers`. For example:

```
{listen,
 [
  ...
  {5280, ejabberd_http, [
                         {request_handlers, [{["http-bind"], mod_http_bind}]},
                         http_poll,
                         web_admin
                        ]
  },
  ...
]}.
```

Options:

`{max_inactivity, Seconds}` Define the maximum inactivity period in seconds. Default value is 30 seconds. For example, to set 50 seconds:

```
{modules,
 [
```

```
    ...
    {mod_http_bind, [ {max_inactivity, 50} ]},
    ...
  ]}.
```

3.3.7 mod_http_fileserver

This simple module serves files from the local disk over HTTP.

Options:

`{docroot, Path}` Directory to serve the files.

`{accesslog, Path}` File to log accesses using an Apache-like format. No log will be recorded if this option is not specified.

`{directory_indices, [Index, ...]}` Indicate one or more directory index files, similarly to Apache's DirectoryIndex variable. When a web request hits a directory instead of a regular file, those directory indices are looked in order, and the first one found is returned.

`{custom_headers, [ {Name, Value}, ...]}` Indicate custom HTTP headers to be included in all responses. Default value is: `[]`

`{content_types, [ {Name, Type}, ...]}` Specify mappings of extension to content type. There are several content types already defined, with this option you can add new definitions, modify or delete existing ones. To delete an existing definition, simply define it with a value: 'undefined'.

`{default_content_type, Type}` Specify the content type to use for unknown extensions. Default value is 'application/octet-stream'.

This example configuration will serve the files from the local directory `/var/www` in the address `http://example.org:5280/pub/archive/`. In this example a new content type `ogg` is defined, `png` is redefined, and `jpg` definition is deleted. To use this module you must enable it:

```
{modules,
 [
  ...
  {mod_http_fileserver, [
                         {docroot, "/var/www"},
                         {accesslog, "/var/log/ejabberd/access.log"},
                         {directory_indices, ["index.html", "main.htm"]},
                         {custom_headers, [{"X-Powered-By", "Erlang/OTP"},
                                           {"X-Fry", "It's a widely-believed fact!"}
                                          ]},
                         {content_types, [{".ogg", "audio/ogg"},
                                          {".png", "image/png"},
                                          {".jpg", undefined}
                                         ]},
```

```
                        {default_content_type, "text/html"}
                       ]
  },
  ...
]}.
```

And define it as a handler in the HTTP service:

```
{listen,
 [
  ...
  {5280, ejabberd_http, [
                         ...
                         {request_handlers, [
                                             ...
                                             {["pub", "archive"], mod_http_fileserver},
                                             ...
                                            ]
                         },
                         ...
                        ]
  },
  ...
]}.
```

3.3.8 mod_irc

This module is an IRC transport that can be used to join channels on IRC servers.

End user information:

- A XMPP client with 'groupchat 1.0' support or Multi-User Chat support (XEP-0045[78]) is necessary to join IRC channels.
- An IRC channel can be joined in nearly the same way as joining a XMPP Multi-User Chat room. The difference is that the room name will be 'channel%`irc.example.org`' in case `irc.example.org` is the IRC server hosting 'channel'. And of course the host should point to the IRC transport instead of the Multi-User Chat service.
- You can register your nickame by sending 'IDENTIFY password' to `nickserver!irc.example.org@irc.jabberserver.org`.
- Entering your password is possible by sending 'LOGIN nick password' to `nickserver!irc.example.org@irc.jabberserver.org`.
- The IRC transport provides Ad-Hoc Commands (XEP-0050[79]) to join a channel, and to set custom IRC username and encoding.

[78] http://xmpp.org/extensions/xep-0045.html
[79] http://xmpp.org/extensions/xep-0050.html

- When using a popular XMPP server, it can occur that no connection can be achieved with some IRC servers because they limit the number of conections from one IP.

Options:

`{host, HostName}` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`irc.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`{access, AccessName}` This option can be used to specify who may use the IRC transport (default value: `all`).

`{default_encoding, Encoding}` Set the default IRC encoding. Default value: `"iso8859-1"`

Examples:

- In the first example, the IRC transport is available on (all) your virtual host(s) with the prefix '`irc.`'. Furthermore, anyone is able to use the transport. The default encoding is set to "iso8859-15".

```
{modules,
 [
  ...
  {mod_irc, [{access, all}, {default_encoding, "iso8859-15"}]},
  ...
 ]}.
```

- In next example the IRC transport is available with JIDs with prefix `irc-t.net`. Moreover, the transport is only accessible to two users of `example.org`, and any user of `example.com`:

```
{acl, paying_customers, {user, "customer1", "example.org"}}.
{acl, paying_customers, {user, "customer2", "example.org"}}.
{acl, paying_customers, {server, "example.com"}}.

{access, irc_users, [{allow, paying_customers}, {deny, all}]}.

{modules,
 [
  ...
  {mod_irc, [{access, irc_users},
             {host, "irc.example.net"}]},
  ...
 ]}.
```

3.3.9 mod_last

This module adds support for Last Activity (XEP-0012[80]). It can be used to discover when a disconnected user last accessed the server, to know when a connected user was last active on the server, or to query the uptime of the `ejabberd` server.

[80] http://xmpp.org/extensions/xep-0012.html

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for Last activity (`jabber:iq:last`) IQ queries (see section 3.3.2).

3.3.10 `mod_muc`

This module provides a Multi-User Chat (XEP-0045[81]) service. Users can discover existing rooms, join or create them. Occupants of a room can chat in public or have private chats.

Some of the features of Multi-User Chat:

- Sending public and private messages to room occupants.
- Inviting other users to a room.
- Setting a room subject.
- Creating password protected rooms.
- Kicking and banning occupants.

The MUC service allows any Jabber ID to register a nickname, so nobody else can use that nickname in any room in the MUC service. To register a nickname, open the Service Discovery in your XMPP client and register in the MUC service.

This module supports clustering and load balancing. One module can be started per cluster node. Rooms are distributed at creation time on all available MUC module instances. The multi-user chat module is clustered but the rooms themselves are not clustered nor fault-tolerant: if the node managing a set of rooms goes down, the rooms disappear and they will be recreated on an available node on first connection attempt.

Module options:

`{host, HostName}` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`conference.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`{access, AccessName}` You can specify who is allowed to use the Multi-User Chat service. By default everyone is allowed to use it.

`{access_create, AccessName}` To configure who is allowed to create new rooms at the Multi-User Chat service, this option can be used. By default any account in the local ejabberd server is allowed to create rooms.

`{access_persistent, AccessName}` To configure who is allowed to modify the 'persistent' room option. By default any account in the local ejabberd server is allowed to modify that option.

[81] http://xmpp.org/extensions/xep-0045.html

`{access_admin, AccessName}` This option specifies who is allowed to administrate the Multi-User Chat service. The default value is `none`, which means that only the room creator can administer his room. The administrators can send a normal message to the service JID, and it will be shown in all active rooms as a service message. The administrators can send a groupchat message to the JID of an active room, and the message will be shown in the room as a service message.

`{history_size, Size}` A small history of the current discussion is sent to users when they enter the room. With this option you can define the number of history messages to keep and send to users joining the room. The value is an integer. Setting the value to `0` disables the history feature and, as a result, nothing is kept in memory. The default value is `20`. This value is global and thus affects all rooms on the service.

`{max_users, Number}` This option defines at the service level, the maximum number of users allowed per room. It can be lowered in each room configuration but cannot be increased in individual room configuration. The default value is 200.

`{max_users_admin_threshold, Number}` This option defines the number of service admins or room owners allowed to enter the room when the maximum number of allowed occupants was reached. The default limit is 5.

`{max_user_conferences, Number}` This option defines the maximum number of rooms that any given user can join. The default value is 10. This option is used to prevent possible abuses. Note that this is a soft limit: some users can sometimes join more conferences in cluster configurations.

`{max_room_id, Number}` This option defines the maximum number of characters that Room ID can have when creating a new room. The default value is to not limit: infinite.

`{max_room_name, Number}` This option defines the maximum number of characters that Room Name can have when configuring the room. The default value is to not limit: infinite.

`{max_room_desc, Number}` This option defines the maximum number of characters that Room Description can have when configuring the room. The default value is to not limit: infinite.

`{min_message_interval, Number}` This option defines the minimum interval between two messages send by an occupant in seconds. This option is global and valid for all rooms. A decimal value can be used. When this option is not defined, message rate is not limited. This feature can be used to protect a MUC service from occupant abuses and limit number of messages that will be broadcasted by the service. A good value for this minimum message interval is 0.4 second. If an occupant tries to send messages faster, an error is send back explaining that the message has been discarded and describing the reason why the message is not acceptable.

`{min_presence_interval, Number}` This option defines the minimum of time between presence changes coming from a given occupant in seconds. This option is global and valid for all rooms. A decimal value can be used. When this option is not defined, no restriction is applied. This option can be used to protect a MUC service for occupants abuses. If an occupant tries to change its presence more often than the specified interval, the presence is cached by `ejabberd` and only the last presence is broadcasted to all occupants in the room after expiration of the interval delay. Intermediate presence packets are silently discarded. A good value for this option is 4 seconds.

`{default_room_options, [ {OptionName, OptionValue}, ...]}` This module option allows to define the desired default room options. Note that the creator of a room can modify the options of his room at any time using a XMPP client with MUC capability. The available room options and the default values are:

`{allow_change_subj, true|false}` Allow occupants to change the subject.

`{allow_private_messages, true|false}` Occupants can send private messages to other occupants.

`{allow_query_users, true|false}` Occupants can send IQ queries to other occupants.

`{allow_user_invites, false|true}` Allow occupants to send invitations.

`{allow_visitor_nickchange, true|false}` Allow visitors to change nickname.

`{allow_visitor_status, true|false}` Allow visitors to send status text in presence updates. If disallowed, the `status` text is stripped before broadcasting the presence update to all the room occupants.

`{anonymous, true|false}` The room is anonymous: occupants don't see the real JIDs of other occupants. Note that the room moderators can always see the real JIDs of the occupants.

`{captcha_protected, false}` When a user tries to join a room where he has no affiliation (not owner, admin or member), the room requires him to fill a CAPTCHA challenge (see section 3.1.8) in order to accept her join in the room.

`{logging, false|true}` The public messages are logged using `mod_muc_log`.

`{max_users, 200}` Maximum number of occupants in the room.

`{members_by_default, true|false}` The occupants that enter the room are participants by default, so they have 'voice'.

`{members_only, false|true}` Only members of the room can enter.

`{moderated, true|false}` Only occupants with 'voice' can send public messages.

`{password, "roompass123"}` Password of the room. You may want to enable the next option too.

`{password_protected, false|true}` The password is required to enter the room.

`{persistent, false|true}` The room persists even if the last participant leaves.

`{public, true|false}` The room is public in the list of the MUC service, so it can be discovered.

`{public_list, true|false}` The list of participants is public, without requiring to enter the room.

`{title, "Room Title"}` A human-readable title of the room.

All of those room options can be set to `true` or `false`, except `password` and `title` which are strings, and `max_users` that is integer.

Examples:

- In the first example everyone is allowed to use the Multi-User Chat service. Everyone will also be able to create new rooms but only the user `admin@example.org` is allowed to administrate any room. In this example he is also a global administrator. When

admin@example.org sends a message such as 'Tomorrow, the XMPP server will be moved to new hardware. This will involve service breakdowns around 23:00 UMT. We apologise for this inconvenience.' to conference.example.org, it will be displayed in all active rooms. In this example the history feature is disabled.

```
{acl, admin, {user, "admin", "example.org"}}.

{access, muc_admin, [{allow, admin}]}.

{modules,
 [
  ...
  {mod_muc, [{access, all},
             {access_create, all},
             {access_admin, muc_admin},
             {history_size, 0}]},
  ...
 ]}.
```

- In the second example the Multi-User Chat service is only accessible by paying customers registered on our domains and on other servers. Of course the administrator is also allowed to access rooms. In addition, he is the only authority able to create and administer rooms. When admin@example.org sends a message such as 'Tomorrow, the Jabber server will be moved to new hardware. This will involve service breakdowns around 23:00 UMT. We apologise for this inconvenience.' to conference.example.org, it will be displayed in all active rooms. No history_size option is used, this means that the feature is enabled and the default value of 20 history messages will be send to the users.

```
{acl, paying_customers, {user, "customer1", "example.net"}}.
{acl, paying_customers, {user, "customer2", "example.com"}}.
{acl, paying_customers, {user, "customer3", "example.org"}}.
{acl, admin, {user, "admin", "example.org"}}.

{access, muc_admin, [{allow, admin},
                     {deny, all}]}.
{access, muc_access, [{allow, paying_customers},
                      {allow, admin},
                      {deny, all}]}.

{modules,
 [
  ...
  {mod_muc, [{access, muc_access},
             {access_create, muc_admin},
             {access_admin, muc_admin}]},
  ...
 ]}.
```

- In the following example, MUC anti abuse options are used. An occupant cannot send more than one message every 0.4 seconds and cannot change its presence more than once

every 4 seconds. The length of Room IDs and Room Names are limited to 20 characters, and Room Description to 300 characters. No ACLs are defined, but some user restriction could be added as well:

```
{modules,
 [
  ...
  {mod_muc, [{min_message_interval, 0.4},
             {min_presence_interval, 4},
             {max_room_id, 20},
             {max_room_name, 20},
             {max_room_desc, 300}]},
  ...
 ]}.
```

- This example shows how to use `default_room_options` to make sure the newly created rooms have by default those options.

```
{modules,
 [
  ...
  {mod_muc, [{access, muc_access},
             {access_create, muc_admin},
             {default_room_options,
              [
               {allow_change_subj, false},
               {allow_query_users, true},
               {allow_private_messages, true},
               {members_by_default, false},
               {title, "New chatroom"},
               {anonymous, false}
              ]},
             {access_admin, muc_admin}]},
  ...
 ]}.
```

3.3.11 `mod_muc_log`

This module enables optional logging of Multi-User Chat (MUC) public conversations to HTML. Once you enable this module, users can join a room using a MUC capable XMPP client, and if they have enough privileges, they can request the configuration form in which they can set the option to enable room logging.

Features:

- Room details are added on top of each page: room title, JID, author, subject and configuration.

- The room JID in the generated HTML is a link to join the room (using XMPP URI[82]).
- Subject and room configuration changes are tracked and displayed.
- Joins, leaves, nick changes, kicks, bans and '/me' are tracked and displayed, including the reason if available.
- Generated HTML files are XHTML 1.0 Transitional and CSS compliant.
- Timestamps are self-referencing links.
- Links on top for quicker navigation: Previous day, Next day, Up.
- CSS is used for style definition, and a custom CSS file can be used.
- URLs on messages and subjects are converted to hyperlinks.
- Timezone used on timestamps is shown on the log files.
- A custom link can be added on top of each page.

Options:

`{access_log, AccessName}` This option restricts which occupants are allowed to enable or disable room logging. The default value is `muc_admin`. Note for this default setting you need to have an access rule for `muc_admin` in order to take effect.

`{cssfile, false|URL}` With this option you can set whether the HTML files should have a custom CSS file or if they need to use the embedded CSS file. Allowed values are `false` and an URL to a CSS file. With the first value, HTML files will include the embedded CSS code. With the latter, you can specify the URL of the custom CSS file (for example: `"http://example.com/my.css"`). The default value is `false`.

`{dirname, room_jid|room_name}` Allows to configure the name of the room directory. Allowed values are `room_jid` and `room_name`. With the first value, the room directory name will be the full room JID. With the latter, the room directory name will be only the room name, not including the MUC service name. The default value is `room_jid`.

`{dirtype, subdirs|plain}` The type of the created directories can be specified with this option. Allowed values are `subdirs` and `plain`. With the first value, subdirectories are created for each year and month. With the latter, the names of the log files contain the full date, and there are no subdirectories. The default value is `subdirs`.

`{file_format, html|plaintext}` Define the format of the log files: `html` stores in HTML format, `plaintext` stores in plain text. The default value is `html`.

`{outdir, Path}` This option sets the full path to the directory in which the HTML files should be stored. Make sure the `ejabberd` daemon user has write access on that directory. The default value is `"www/muc"`.

`{spam_prevention true|false}` To prevent spam, the `spam_prevention` option adds a special attribute to links that prevent their indexation by search engines. The default value is `true`, which mean that nofollow attributes will be added to user submitted links.

[82] http://xmpp.org/rfcs/rfc5122.html

`{timezone, local|universal}` The time zone for the logs is configurable with this option. Allowed values are `local` and `universal`. With the first value, the local time, as reported to Erlang by the operating system, will be used. With the latter, GMT/UTC time will be used. The default value is `local`.

`{top_link, {URL, Text}}` With this option you can customize the link on the top right corner of each log file. The default value is {`"/"`, `"Home"`}.

Examples:

- In the first example any room owner can enable logging, and a custom CSS file will be used (http://example.com/my.css). The names of the log files will contain the full date, and there will be no subdirectories. The log files will be stored in /var/www/muclogs, and the time zone will be GMT/UTC. Finally, the top link will be `<a href="http://www.jabber.ru/">Jabber.ru</a>`.

```
{access, muc, [{allow, all}]}.

{modules,
 [
  ...
  {mod_muc_log, [
                 {access_log, muc},
                 {cssfile, "http://example.com/my.css"},
                 {dirtype, plain},
                 {dirname, room_jid},
                 {outdir, "/var/www/muclogs"},
                 {timezone, universal},
                 {spam_prevention, true},
                 {top_link, {"http://www.jabber.ru/", "Jabber.ru"}}
                ]},
  ...
 ]}.
```

- In the second example only `admin1@example.org` and `admin2@example.net` can enable logging, and the embedded CSS file will be used. The names of the log files will only contain the day (number), and there will be subdirectories for each year and month. The log files will be stored in /var/www/muclogs, and the local time will be used. Finally, the top link will be the default `<a href="/">Home</a>`.

```
{acl, admins, {user, "admin1", "example.org"}}.
{acl, admins, {user, "admin2", "example.net"}}.

{access, muc_log, [{allow, admins},
                   {deny, all}]}.

{modules,
 [
  ...
  {mod_muc_log, [
```

```
                    {access_log, muc_log},
                    {cssfile, false},
                    {dirtype, subdirs},
                    {outdir, "/var/www/muclogs"},
                    {timezone, local}
                   ]},
     ...
    ]}.
```

3.3.12 mod_offline

This module implements offline message storage (XEP-0160[83]). This means that all messages sent to an offline user will be stored on the server until that user comes online again. Thus it is very similar to how email works. Note that `ejabberdctl` has a command to delete expired messages (see section 4.1).

`{access_max_user_messages, AccessName}` This option defines which access rule will be enforced to limit the maximum number of offline messages that a user can have (quota). When a user has too many offline messages, any new messages that he receive are discarded, and a resource-constraint error is returned to the sender. The default value is `max_user_offline_messages`. Then you can define an access rule with a syntax similar to `max_user_sessions` (see 3.1.5).

This example allows power users to have as much as 5000 offline messages, administrators up to 2000, and all the other users up to 100.

```
{acl, admin, {user, "admin1", "localhost"}}.
{acl, admin, {user, "admin2", "example.org"}}.
{acl, poweruser, {user, "bob", "example.org"}}.
{acl, poweruser, {user, "jane", "example.org"}}.

{access, max_user_offline_messages, [ {5000, poweruser}, {2000, admin}, {100, all} ]}.

{modules,
 [
  ...
  {mod_offline,  [ {access_max_user_messages, max_user_offline_messages} ]},
  ...
 ]}.
```

3.3.13 mod_ping

This module implements support for XMPP Ping (XEP-0199[84]) and periodic keepalives. When this module is enabled ejabberd responds correctly to ping requests, as defined in the protocol.

[83] http://xmpp.org/extensions/xep-0160.html
[84] http://xmpp.org/extensions/xep-0199.html

Configuration options:

`{send_pings, true|false}` If this option is set to `true`, the server sends pings to connected clients that are not active in a given interval `ping_interval`. This is useful to keep client connections alive or checking availability. By default this option is disabled.

`{ping_interval, Seconds}` How often to send pings to connected clients, if the previous option is enabled. If a client connection does not send or receive any stanza in this interval, a ping request is sent to the client. The default value is 60 seconds.

`{timeout_action, none|kill}` What to do when a client does not answer to a server ping request in less than 32 seconds. The default is to do nothing.

This example enables Ping responses, configures the module to send pings to client connections that are inactive for 4 minutes, and if a client does not answer to the ping in less than 32 seconds, its connection is closed:

```
{modules,
 [
  ...
  {mod_ping,  [{send_pings, true}, {ping_interval, 240}, {timeout_action, kill}]},
  ...
 ]}.
```

3.3.14 mod_privacy

This module implements Blocking Communication (also known as Privacy Rules) as defined in section 10 from XMPP IM. If end users have support for it in their XMPP client, they will be able to:

- Retrieving one's privacy lists.
- Adding, removing, and editing one's privacy lists.
- Setting, changing, or declining active lists.
- Setting, changing, or declining the default list (i.e., the list that is active by default).
- Allowing or blocking messages based on JID, group, or subscription type (or globally).
- Allowing or blocking inbound presence notifications based on JID, group, or subscription type (or globally).
- Allowing or blocking outbound presence notifications based on JID, group, or subscription type (or globally).
- Allowing or blocking IQ stanzas based on JID, group, or subscription type (or globally).

- Allowing or blocking all communications based on JID, group, or subscription type (or globally).

(from http://xmpp.org/rfcs/rfc3921.html#privacy)

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for Blocking Communication (`jabber:iq:privacy`) IQ queries (see section 3.3.2).

3.3.15 `mod_private`

This module adds support for Private XML Storage (XEP-0049[85]):

> Using this method, XMPP entities can store private data on the server and retrieve it whenever necessary. The data stored might be anything, as long as it is valid XML. One typical usage for this namespace is the server-side storage of client-specific preferences; another is Bookmark Storage (XEP-0048[86]).

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for Private XML Storage (`jabber:iq:private`) IQ queries (see section 3.3.2).

3.3.16 `mod_proxy65`

This module implements SOCKS5 Bytestreams (XEP-0065[87]). It allows `ejabberd` to act as a file transfer proxy between two XMPP clients.

Options:

`{host, HostName}` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`proxy.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`{name, Text}` Defines Service Discovery name of the service. Default is `"SOCKS5 Bytestreams"`.

`{ip, IPTuple}` This option specifies which network interface to listen for. Default is an IP address of the service's DNS name, or, if fails, `{127,0,0,1}`.

`{port, Number}` This option defines port to listen for incoming connections. Default is 7777.

[85] http://xmpp.org/extensions/xep-0049.html
[86] http://xmpp.org/extensions/xep-0048.html
[87] http://xmpp.org/extensions/xep-0065.html

`{hostname, HostName}` Defines a hostname advertised by the service when establishing a session with clients. This is useful when you run the service behind a NAT. The default is the value of `ip` option. Examples: `"proxy.mydomain.org"`, `"200.150.100.50"`. Note that not all clients understand domain names in stream negotiation, so you should think twice before setting domain name in this option.

`{auth_type, anonymous|plain}` SOCKS5 authentication type. Possible values are `anonymous` and `plain`. Default is `anonymous`.

`{access, AccessName}` Defines ACL for file transfer initiators. Default is `all`.

`{max_connections, Number}` Maximum number of active connections per file transfer initiator. No limit by default.

`{shaper, none|ShaperName}` This option defines shaper for the file transfer peers. Shaper with the maximum bandwidth will be selected. Default is `none`.

Examples:

- The simpliest configuration of the module:

```
{modules,
 [
  ...
  {mod_proxy65, []},
  ...
 ]}.
```

- More complicated configuration.

```
{acl, proxy_users, {server, "example.org"}}.
{access, proxy65_access, [{allow, proxy_users}, {deny, all}]}.

{acl, admin, {user, "admin", "example.org"}}.
{shaper, proxyrate, {maxrate, 10240}}. %% 10 Kbytes/sec
{access, proxy65_shaper, [{none, admin}, {proxyrate, proxy_users}]}.

{modules,
 [
  ...
  {mod_proxy65, [{host, "proxy1.example.org"},
                 {name, "File Transfer Proxy"},
                 {ip, {200,150,100,1}},
                 {port, 7778},
                 {max_connections, 5},
                 {access, proxy65_access},
                 {shaper, proxy65_shaper}]},
  ...
 ]}.
```

3.3.17 mod_pubsub

This module offers a Publish-Subscribe Service (XEP-0060[88]). The functionality in `mod_pubsub` can be extended using plugins. The plugin that implements PEP (Personal Eventing via Pubsub) (XEP-0163[89]) is enabled in the default ejabberd configuration file, and it requires `mod_caps`.

Options:

`{host, HostName}` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`pubsub.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name. If you use `mod_pubsub_odbc`, please ensure the prefix contains only one dot, for example '`pubsub.`', or '`publish.`',.

`{access_createnode, AccessName}` This option restricts which users are allowed to create pubsub nodes using ACL and ACCESS. By default any account in the local ejabberd server is allowed to create pubsub nodes.

`{max_items_node, MaxItems}` Define the maximum number of items that can be stored in a node. Default value is 10.

`{plugins, [ Plugin, ...]}` To specify which pubsub node plugins to use. The first one in the list is used by default. If this option is not defined, the default plugins list is: `["flat"]`. PubSub clients can define which plugin to use when creating a node: add `type='plugin-name'` attribute to the `create` stanza element.

`{nodetree, Nodetree}` To specify which nodetree to use. If not defined, the default pubsub nodetree is used: "tree". Only one nodetree can be used per host, and is shared by all node plugins.

The "virtual" nodetree does not store nodes on database. This saves resources on systems with tons of nodes. If using the "virtual" nodetree, you can only enable those node plugins: ["flat","pep"] or ["flat"]; any other plugins configuration will not work. Also, all nodes will have the defaut configuration, and this can not be changed. Using "virtual" nodetree requires to start from a clean database, it will not work if you used the default "tree" nodetree before.

The "dag" nodetree provides experimental support for PubSub Collection Nodes (XEP-0248[90]). In that case you should also add "dag" node plugin as default, for example: `{plugins, ["dag","flat","hometree","pep"]}`

`{ignore_pep_from_offline, false|true}` To specify whether or not we should get last published PEP items from users in our roster which are offline when we connect. Value is true or false. If not defined, pubsub assumes true so we only get last items of online contacts.

`{last_item_cache, false|true}` To specify whether or not pubsub should cache last items. Value is true or false. If not defined, pubsub do not cache last items. On systems with not so many nodes, caching last items speeds up pubsub and allows to raise user connection rate. The cost is memory usage, as every item is stored in memory.

[88] http://xmpp.org/extensions/xep-0060.html
[89] http://xmpp.org/extensions/xep-0163.html
[90] http://xmpp.org/extensions/xep-0248.html

`{pep_mapping, [ {Key, Value}, ...]}` This allow to define a Key-Value list to choose defined node plugins on given PEP namespace. The following example will use node_tune instead of node_pep for every PEP node with tune namespace:

```
{mod_pubsub, [{pep_mapping, [{"http://jabber.org/protocol/tune", "tune"}]}]}
```

Example of configuration that uses flat nodes as default, and allows use of flat, nodetree and pep nodes:

```
{modules,
 [
  ...
  {mod_pubsub, [
                {access_createnode, pubsub_createnode},
                {plugins, ["flat", "hometree", "pep"]}
               ]},
  ...
 ]}.
```

Using ODBC database requires use of dedicated plugins. The following example shows previous configuration with ODBC usage:

```
{modules,
 [
  ...
  {mod_pubsub_odbc, [
                {access_createnode, pubsub_createnode},
                {plugins, ["flat_odbc", "hometree_odbc", "pep_odbc"]}
               ]},
  ...
 ]}.
```

3.3.18 mod_register

This module adds support for In-Band Registration (XEP-0077[91]). This protocol enables end users to use a XMPP client to:

- Register a new account on the server.
- Change the password from an existing account on the server.
- Delete an existing account on the server.

Options:

[91] http://xmpp.org/extensions/xep-0077.html

`{access, AccessName}` This option can be configured to specify rules to restrict registration. If a rule returns 'deny' on the requested user name, registration for that user name is denied. (there are no restrictions by default).

`{access_from, AccessName}` By default, `ejabberd` doesn't allow to register new accounts from s2s or existing c2s sessions. You can change it by defining access rule in this option. Use with care: allowing registration from s2s leads to uncontrolled massive accounts creation by rogue users.

`{captcha_protected, false|true}` Protect registrations with CAPTCHA (see section 3.1.8). The default is `false`.

`{ip_access, [ {allow|deny, IPaddress}, ...]}` Define rules to allow or deny account registration depending in the IP address of the XMPP client. If there is no matching IP mask, the default rule is "allow". IPv6 addresses are supported, but not tested. The default option value is an empty list: `[]`.

`{password_strength, Entropy}` This option sets the minimum informational entropy for passwords. The value `Entropy` is a number of bits of entropy. The recommended minimum is 32 bits. The default is 0, i.e. no checks are performed.

`{welcome_message, Message}` Set a welcome message that is sent to each newly registered account. The first string is the subject, and the second string is the message body. In the body you can set a newline with the characters: `\n`

`{registration_watchers, [ JID, ...]}` This option defines a list of JIDs which will be notified each time a new account is registered.

`{iqdisc, Discipline}` This specifies the processing discipline for In-Band Registration (`jabber:iq:register`) IQ queries (see section 3.3.2).

This module reads also another option defined globally for the server: `{registration_timeout, Timeout}`. This option limits the frequency of registration from a given IP or username. So, a user that tries to register a new account from the same IP address or JID during this number of seconds after his previous registration will receive an error `resource-constraint` with the explanation: "Users are not allowed to register accounts so quickly". The timeout is expressed in seconds, and it must be an integer. To disable this limitation, instead of an integer put a word like: `infinity`. Default value: 600 seconds.

Examples:

- Next example prohibits the registration of too short account names, and allows to create accounts only to clients of the local network:

```
{acl, shortname, {user_glob, "?"}}.
{acl, shortname, {user_glob, "??"}}.
%% The same using regexp:
%%{acl, shortname, {user_regexp, "^..?$"}}.

{access, register, [{deny, shortname},
                    {allow, all}]}.
```

```
{modules,
 [
  ...
  {mod_register, [{access, register},
                  {ip_access, [{allow, "127.0.0.0/8"},
                               {deny, "0.0.0.0/0"}]}
  ]},
  ...
 ]}.
```

- This configuration prohibits usage of In-Band Registration to create or delete accounts, but allows existing accounts to change the password:

```
{access, register, [{deny, all}]}.

{modules,
 [
  ...
  {mod_register, [{access, register}]},
  ...
 ]}.
```

- This configuration disables all In-Band Registration functionality: create, delete accounts and change password:

```
{modules,
 [
  ...
  %% {mod_register, [{access, register}]},
  ...
 ]}.
```

- Define the welcome message and two registration watchers. Also define a registration timeout of one hour:

```
{registration_timeout, 3600}.
{modules,
 [
  ...
  {mod_register,
   [
    {welcome_message, {"Welcome!", "Hi.\nWelcome to this Jabber server.\n Check http://www.jabbe
    {registration_watchers, ["admin1@example.org", "boss@example.net"]}
   ]},
  ...
 ]}.
```

3.3.19 mod_register_web

This module provides a web page where people can:

- Register a new account on the server.
- Change the password from an existing account on the server.
- Delete an existing account on the server.

This module supports CAPTCHA image to register a new account. To enable this feature, configure the options captcha_cmd and captcha_host.

Options:

`{registration_watchers, [ JID, ...]}` This option defines a list of JIDs which will be notified each time a new account is registered.

This example configuration shows how to enable the module and the web handler:

```
{listen, [
  ...
  {5281, ejabberd_http, [
    tls,
    {certfile, "/etc/ejabberd/certificate.pem"},
    register
  ]},
  ...
]}.

{modules,
 [
  ...
  {mod_register_web, []},
  ...
 ]}.
```

The users can visit this page: https://localhost:5281/register/ It is important to include the last / character in the URL, otherwise the subpages URL will be incorrect.

3.3.20 mod_roster

This module implements roster management as defined in RFC 3921: XMPP IM[92]. It also supports Roster Versioning (XEP-0237[93]).

Options:

[92] http://xmpp.org/rfcs/rfc3921.html#roster
[93] http://xmpp.org/extensions/xep-0237.html

`{iqdisc, Discipline}` This specifies the processing discipline for Roster Management (`jabber:iq:roster`) IQ queries (see section 3.3.2).

`{versioning, false|true}` Enables Roster Versioning. This option is disabled by default.

`{store_current_id, false|true}` If this option is enabled, the current version number is stored on the database. If disabled, the version number is calculated on the fly each time. Enabling this option reduces the load for both ejabberd and the database. This option does not affect the client in any way. This option is only useful if Roster Versioning is enabled. This option is disabled by default. Important: if you use `mod_shared_roster` or `mod_shared_roster_ldap`, you must disable this option.

This example configuration enables Roster Versioning with storage of current id:

```
{modules,
 [
  ...
  {mod_roster, [{versioning, true}, {store_current_id, true}]},
  ...
 ]}.
```

3.3.21 mod_service_log

This module adds support for logging end user packets via a XMPP message auditing service such as Bandersnatch[94]. All user packets are encapsulated in a `<route/>` element and sent to the specified service(s).

Options:

`{loggers, [Names, ...]}` With this option a (list of) service(s) that will receive the packets can be specified.

Examples:

- To log all end user packets to the Bandersnatch service running on `bandersnatch.example.com`:

  ```
  {modules,
   [
    ...
    {mod_service_log, [{loggers, ["bandersnatch.example.com"]}]},
    ...
   ]}.
  ```

- To log all end user packets to the Bandersnatch service running on `bandersnatch.example.com` and the backup service on `bandersnatch.example.org`:

[94] http://www.funkypenguin.info/project/bandersnatch/

```
{modules,
 [
  ...
  {mod_service_log, [{loggers, ["bandersnatch.example.com",
                                "bandersnatch.example.org"]}]},
  ...
 ]}.
```

3.3.22 mod_shared_roster

This module enables you to create shared roster groups. This means that you can create groups of people that can see members from (other) groups in their rosters. The big advantages of this feature are that end users do not need to manually add all users to their rosters, and that they cannot permanently delete users from the shared roster groups. A shared roster group can have members from any XMPP server, but the presence will only be available from and to members of the same virtual host where the group is created.

Shared roster groups can be edited *only* via the Web Admin. Each group has a unique identification and the following parameters:

Name The name of the group, which will be displayed in the roster.

Description The description of the group. This parameter does not affect anything.

Members A list of JIDs of group members, entered one per line in the Web Admin. The special member directive `@all@` represents all the registered users in the virtual host; which is only recommended for a small server with just a few hundred users. The special member directive `@online@` represents the online users in the virtual host.

Displayed groups A list of groups that will be in the rosters of this group's members.

Examples:

- Take the case of a computer club that wants all its members seeing each other in their rosters. To achieve this, they need to create a shared roster group similar to next table:

Identification	Group 'club_members'
Name	Club Members
Description	Members from the computer club
Members	member1@example.org member2@example.org member3@example.org
Displayed groups	club_members

- In another case we have a company which has three divisions: Management, Marketing and Sales. All group members should see all other members in their rosters. Additionally, all managers should have all marketing and sales people in their roster. Simultaneously,

all marketeers and the whole sales team should see all managers. This scenario can be achieved by creating shared roster groups as shown in the following table:

Identification	Group 'management'	Group 'marketing'	Group 'sales'
Name	Management	Marketing	Sales
Description			
Members	manager1@example.org manager2@example.org manager3@example.org manager4@example.org	marketeer1@example.org marketeer2@example.org marketeer3@example.org marketeer4@example.org	saleswoman1@example.org salesman1@example.org saleswoman2@example.org salesman2@example.org
Displayed groups	management marketing sales	management marketing	management sales

3.3.23 mod_shared_roster_ldap

This module lets the server administrator automatically populate users' rosters (contact lists) with entries based on users and groups defined in an LDAP-based directory.

Configuration parameters

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

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 *must* 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))"
```

Attributes

These parameters specify the names of the attributes which hold interesting data in the entries returned by running filters specified in section 3.3.23.

`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`.

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 3.3.23.

`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 3.3.23.

`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 3.3.23 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 3.3.23 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 3.3.23 on how it is used during roster retrieval.

Connection parameters

The module also accepts the connection parameters, all of which default to the top-level parameter of the same name, if unspecified. See 3.2.5 for more information about them.

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`.
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.

Configuration examples

Since there are many possible DIT[95] 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).

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.

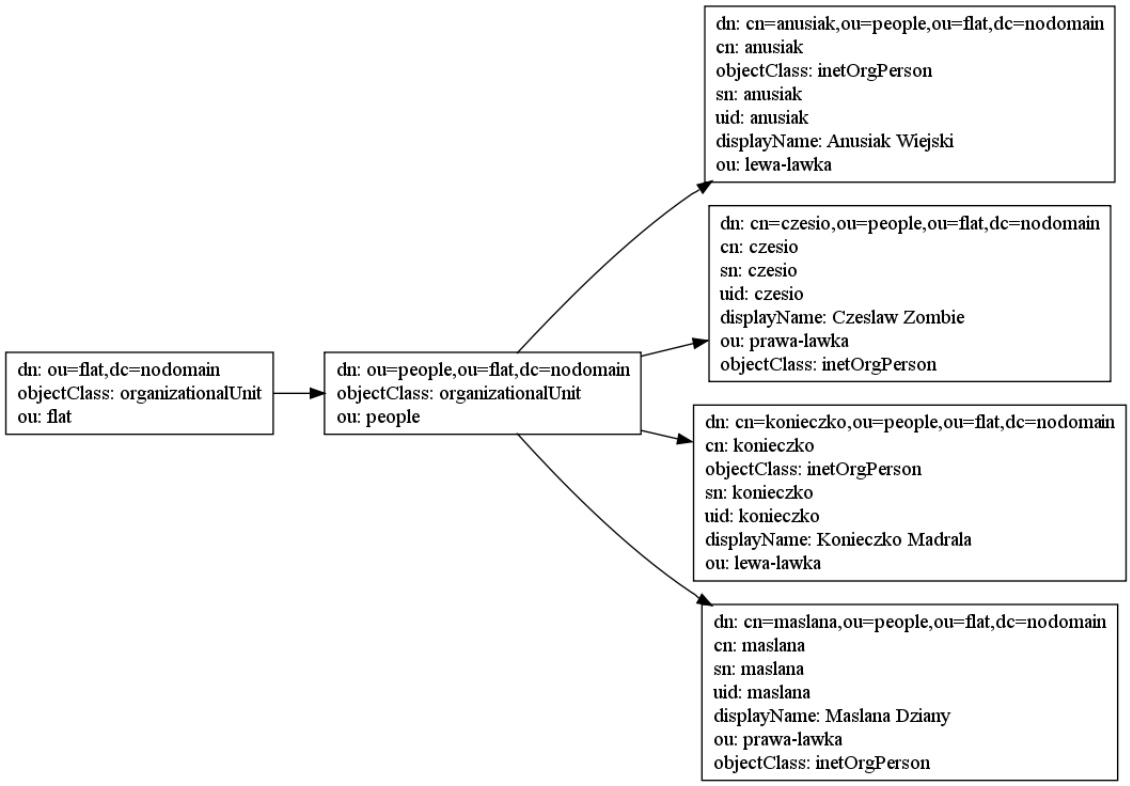

Figure 3.1: Flat DIT graph

Such layout has a few downsides, including:

- 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

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

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`.

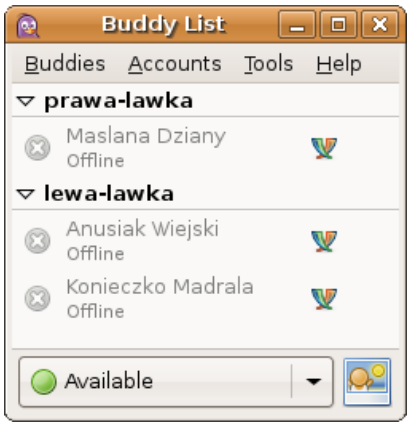

Figure 3.2: Roster from flat DIT

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)"},
  {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.

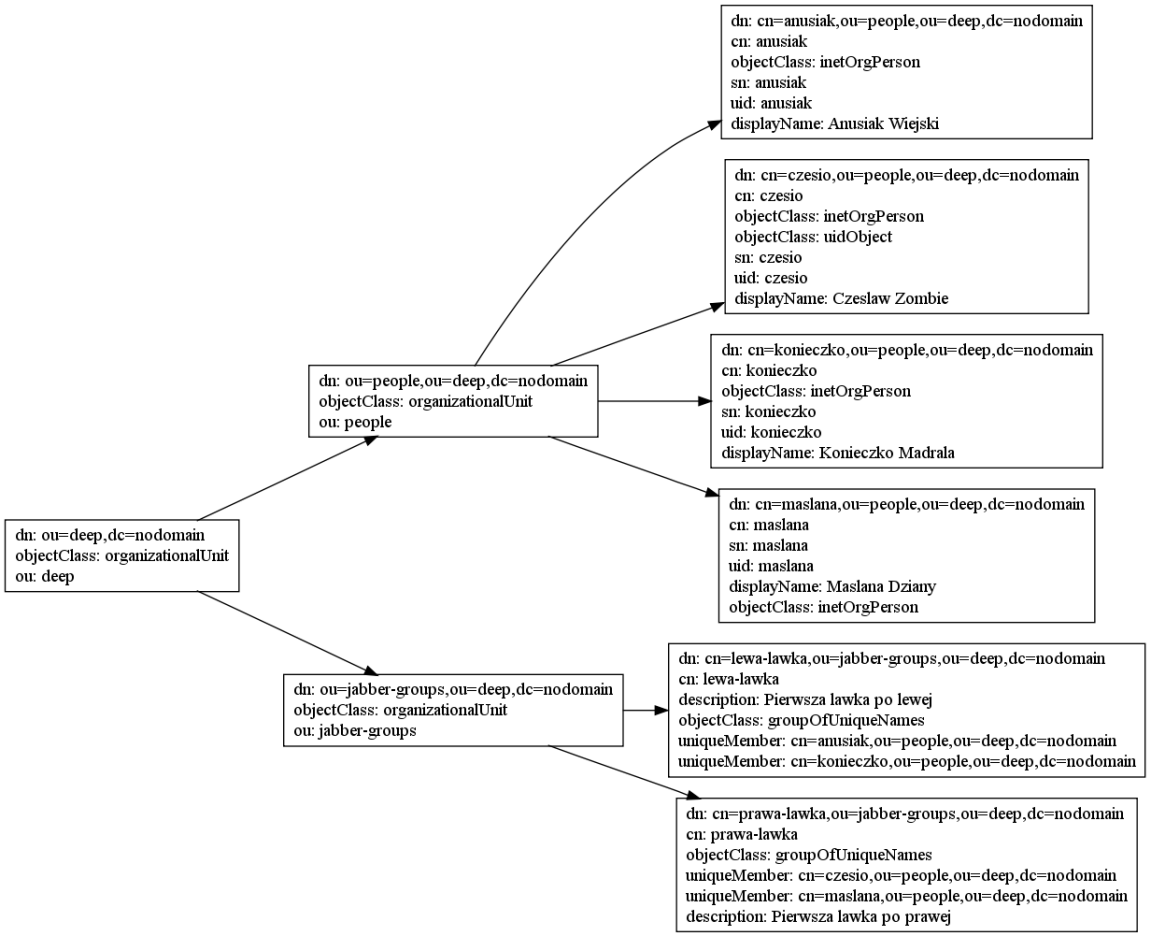

Figure 3.3: Example “deep” DIT graph

Figure 3.4: Example roster from “deep” DIT

3.3.24 mod_sic

This module adds support for Server IP Check (XEP-0279[96]). This protocol enables a client to discover its external IP address.

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for `urn:xmpp:sic:0` IQ queries (see section 3.3.2).

3.3.25 mod_stats

This module adds support for Statistics Gathering (XEP-0039[97]). This protocol allows you to retrieve next statistics from your `ejabberd` deployment:

- Total number of registered users on the current virtual host (users/total).
- Total number of registered users on all virtual hosts (users/all-hosts/total).
- Total number of online users on the current virtual host (users/online).
- Total number of online users on all virtual hosts (users/all-hosts/online).

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for Statistics Gathering (`http://jabber.org/pro`
IQ queries (see section 3.3.2).

As there are only a small amount of clients (for example Tkabber[98]) and software libraries with support for this XEP, a few examples are given of the XML you need to send in order to get the statistics. Here they are:

- You can request the number of online users on the current virtual host (`example.org`) by sending:

```
<iq to='example.org' type='get'>
  <query xmlns='http://jabber.org/protocol/stats'>
    <stat name='users/online'/>
  </query>
</iq>
```

- You can request the total number of registered users on all virtual hosts by sending:

```
<iq to='example.org' type='get'>
  <query xmlns='http://jabber.org/protocol/stats'>
    <stat name='users/all-hosts/total'/>
  </query>
</iq>
```

[96] http://xmpp.org/extensions/xep-0279.html
[97] http://xmpp.org/extensions/xep-0039.html
[98] http://tkabber.jabber.ru/

3.3.26 mod_time

This module features support for Entity Time (XEP-0202[99]). By using this XEP, you are able to discover the time at another entity's location.

Options:

`{iqdisc, Discipline}` This specifies the processing discipline for Entity Time (`jabber:iq:time`) IQ queries (see section 3.3.2).

3.3.27 mod_vcard

This module allows end users to store and retrieve their vCard, and to retrieve other users vCards, as defined in vcard-temp (XEP-0054[100]). The module also implements an uncomplicated Jabber User Directory based on the vCards of these users. Moreover, it enables the server to send its vCard when queried.

Options:

`{host, HostName}` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`vjud.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`{iqdisc, Discipline}` This specifies the processing discipline for `vcard-temp` IQ queries (see section 3.3.2).

`{search, true|false}` This option specifies whether the search functionality is enabled or not If disabled, the option `host` will be ignored and the Jabber User Directory service will not appear in the Service Discovery item list. The default value is `true`.

`{matches, infinity|Number}` With this option, the number of reported search results can be limited. If the option's value is set to `infinity`, all search results are reported. The default value is `30`.

`{allow_return_all, false|true}` This option enables you to specify if search operations with empty input fields should return all users who added some information to their vCard. The default value is `false`.

`{search_all_hosts, true|false}` If this option is set to `true`, search operations will apply to all virtual hosts. Otherwise only the current host will be searched. The default value is `true`. This option is available in `mod_vcard`, but not available in `mod_vcard_odbc`.

Examples:

- In this first situation, search results are limited to twenty items, every user who added information to their vCard will be listed when people do an empty search, and only users from the current host will be returned:

[99] http://xmpp.org/extensions/xep-0202.html
[100] http://xmpp.org/extensions/xep-0054.html

```
{modules,
 [
  ...
  {mod_vcard, [{search, true},
               {matches, 20},
               {allow_return_all, true},
               {search_all_hosts, false}]},
  ...
 ]}.
```

- The second situation differs in a way that search results are not limited, and that all virtual hosts will be searched instead of only the current one:

```
{modules,
 [
  ...
  {mod_vcard, [{search, true},
               {matches, infinity},
               {allow_return_all, true}]},
  ...
 ]}.
```

3.3.28 mod_vcard_ldap

ejabberd can map LDAP attributes to vCard fields. This behaviour is implemented in the mod_vcard_ldap module. This module does not depend on the authentication method (see 3.2.5).

Usually ejabberd treats LDAP as a read-only storage: it is possible to consult data, but not possible to create accounts or edit vCard that is stored in LDAP. However, it is possible to change passwords if mod_register module is enabled and LDAP server supports RFC 3062[101].

The mod_vcard_ldap module has its own optional parameters. The first group of parameters has the same meaning as the top-level LDAP parameters to set the authentication method: ldap_servers, ldap_port, ldap_rootdn, ldap_password, ldap_base, ldap_uids, and ldap_filter. See section 3.2.5 for detailed information about these options. If one of these options is not set, ejabberd will look for the top-level option with the same name.

The second group of parameters consists of the following mod_vcard_ldap-specific options:

`{host, HostName}` This option defines the Jabber ID of the service. If the host option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix 'vjud.'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`{iqdisc, Discipline}` This specifies the processing discipline for vcard-temp IQ queries (see section 3.3.2).

[101] http://tools.ietf.org/html/rfc3062

`{search, true|false}` This option specifies whether the search functionality is enabled (value: `true`) or disabled (value: `false`). If disabled, the option `host` will be ignored and the Jabber User Directory service will not appear in the Service Discovery item list. The default value is `true`.

`{matches, infinity|Number}` With this option, the number of reported search results can be limited. If the option's value is set to `infinity`, all search results are reported. The default value is `30`.

`{ldap_vcard_map, [ {Name, Pattern, LDAPattributes}, ...]}` With this option you can set the table that maps LDAP attributes to vCard fields. `Name` is the type name of the vCard as defined in RFC 2426[102]. `Pattern` is a string which contains pattern variables `"%u"`, `"%d"` or `"%s"`. `LDAPattributes` is the list containing LDAP attributes. The pattern variables `"%s"` will be sequentially replaced with the values of LDAP attributes from `List_of_LDAP_attributes`, `"%u"` will be replaced with the user part of a JID, and `"%d"` will be replaced with the domain part of a JID. The default is:

```
[{"NICKNAME", "%u", []},
 {"FN", "%s", ["displayName"]},
 {"LAST", "%s", ["sn"]},
 {"FIRST", "%s", ["givenName"]},
 {"MIDDLE", "%s", ["initials"]},
 {"ORGNAME", "%s", ["o"]},
 {"ORGUNIT", "%s", ["ou"]},
 {"CTRY", "%s", ["c"]},
 {"LOCALITY", "%s", ["l"]},
 {"STREET", "%s", ["street"]},
 {"REGION", "%s", ["st"]},
 {"PCODE", "%s", ["postalCode"]},
 {"TITLE", "%s", ["title"]},
 {"URL", "%s", ["labeleduri"]},
 {"DESC", "%s", ["description"]},
 {"TEL", "%s", ["telephoneNumber"]},
 {"EMAIL", "%s", ["mail"]},
 {"BDAY", "%s", ["birthDay"]},
 {"ROLE", "%s", ["employeeType"]},
 {"PHOTO", "%s", ["jpegPhoto"]}]
```

`{ldap_search_fields, [ {Name, Attribute}, ...]}` This option defines the search form and the LDAP attributes to search within. `Name` is the name of a search form field which will be automatically translated by using the translation files (see `msgs/*.msg` for available words). `Attribute` is the LDAP attribute or the pattern `"%u"`. The default is:

```
[{"User", "%u"},
 {"Full Name", "displayName"},
 {"Given Name", "givenName"},
 {"Middle Name", "initials"},
 {"Family Name", "sn"},
 {"Nickname", "%u"},
```

[102] http://tools.ietf.org/html/rfc2426

```
 {"Birthday", "birthDay"},
 {"Country", "c"},
 {"City", "l"},
 {"Email", "mail"},
 {"Organization Name", "o"},
 {"Organization Unit", "ou"}]
```

`{ldap_search_reported, [ {SearchField, VcardField}, ...]}` This option defines which search fields should be reported. `SearchField` is the name of a search form field which will be automatically translated by using the translation files (see `msgs/*.msg` for available words). `VcardField` is the vCard field name defined in the `ldap_vcard_map` option. The default is:

```
[{"Full Name", "FN"},
 {"Given Name", "FIRST"},
 {"Middle Name", "MIDDLE"},
 {"Family Name", "LAST"},
 {"Nickname", "NICKNAME"},
 {"Birthday", "BDAY"},
 {"Country", "CTRY"},
 {"City", "LOCALITY"},
 {"Email", "EMAIL"},
 {"Organization Name", "ORGNAME"},
 {"Organization Unit", "ORGUNIT"}]
```

Examples:

- Let's say `ldap.example.org` is the name of our LDAP server. We have users with their passwords in `"ou=Users,dc=example,dc=org"` directory. Also we have addressbook, which contains users emails and their additional infos in `"ou=AddressBook,dc=example,dc=org"` directory. Corresponding authentication section should looks like this:

  ```
  %% authentication method
  {auth_method, ldap}.
  %% DNS name of our LDAP server
  {ldap_servers, ["ldap.example.org"]}.
  %% We want to authorize users from 'shadowAccount' object class only
  {ldap_filter, "(objectClass=shadowAccount)"}.
  ```

  Now we want to use users LDAP-info as their vCards. We have four attributes defined in our LDAP schema: `"mail"` — email address, `"givenName"` — first name, `"sn"` — second name, `"birthDay"` — birthday. Also we want users to search each other. Let's see how we can set it up:

  ```
  {modules,
    ...
    {mod_vcard_ldap,
     [
  ```

```
    %% We use the same server and port, but want to bind anonymously because
    %% our LDAP server accepts anonymous requests to
    %% "ou=AddressBook,dc=example,dc=org" subtree.
    {ldap_rootdn, ""},
    {ldap_password, ""},
    %% define the addressbook's base
    {ldap_base, "ou=AddressBook,dc=example,dc=org"},
    %% uidattr: user's part of JID is located in the "mail" attribute
    %% uidattr_format: common format for our emails
    {ldap_uids, [{"mail","%u@mail.example.org"}]},
    %% We have to define empty filter here, because entries in addressbook does not
    %% belong to shadowAccount object class
    {ldap_filter, ""},
    %% Now we want to define vCard pattern
    {ldap_vcard_map,
     [{"NICKNAME", "%u", []}, % just use user's part of JID as his nickname
      {"FIRST", "%s", ["givenName"]},
      {"LAST", "%s", ["sn"]},
      {"FN", "%s, %s", ["sn", "givenName"]}, % example: "Smith, John"
      {"EMAIL", "%s", ["mail"]},
      {"BDAY", "%s", ["birthDay"]}]},
    %% Search form
    {ldap_search_fields,
     [{"User", "%u"},
      {"Name", "givenName"},
      {"Family Name", "sn"},
      {"Email", "mail"},
      {"Birthday", "birthDay"}]},
    %% vCard fields to be reported
    %% Note that JID is always returned with search results
    {ldap_search_reported,
     [{"Full Name", "FN"},
      {"Nickname", "NICKNAME"},
      {"Birthday", "BDAY"}]}
  ]}
  ...
}.
```

Note that `mod_vcard_ldap` module checks an existence of the user before searching his info in LDAP.

- `ldap_vcard_map` example:

```
{ldap_vcard_map,
 [{"NICKNAME", "%u", []},
  {"FN", "%s", ["displayName"]},
  {"CTRY", "Russia", []},
  {"EMAIL", "%u@%d", []},
  {"DESC", "%s\n%s", ["title", "description"]}
 ]},
```

- `ldap_search_fields` example:

```
{ldap_search_fields,
 [{"User", "uid"},
  {"Full Name", "displayName"},
  {"Email", "mail"}
 ]},
```

- `ldap_search_reported` example:

```
{ldap_search_reported,
 [{"Full Name", "FN"},
  {"Email", "EMAIL"},
  {"Birthday", "BDAY"},
  {"Nickname", "NICKNAME"}
 ]},
```

3.3.29 mod_vcard_xupdate

The user's client can store an avatar in the user vCard. The vCard-Based Avatars protocol (XEP-0153[103]) provides a method for clients to inform the contacts what is the avatar hash value. However, simple or small clients may not implement that protocol.

If this module is enabled, all the outgoing client presence stanzas get automatically the avatar hash on behalf of the client. So, the contacts receive the presence stanzas with the Update Data described in XEP-0153[104] as if the client would had inserted it itself. If the client had already included such element in the presence stanza, it is replaced with the element generated by ejabberd.

By enabling this module, each vCard modification produces a hash recalculation, and each presence sent by a client produces hash retrieval and a presence stanza rewrite. For this reason, enabling this module will introduce a computational overhead in servers with clients that change frequently their presence.

3.3.30 mod_version

This module implements Software Version (XEP-0092[105]). Consequently, it answers `ejabberd`'s version when queried.

Options:

`{show_os, true|false}` Should the operating system be revealed or not. The default value is `true`.

`{iqdisc, Discipline}` This specifies the processing discipline for Software Version (`jabber:iq:version`) IQ queries (see section 3.3.2).

[103] http://xmpp.org/extensions/xep-0153.html
[104] http://xmpp.org/extensions/xep-0153.html
[105] http://xmpp.org/extensions/xep-0092.html

Chapter 4

Managing an ejabberd Server

4.1 ejabberdctl

With the `ejabberdctl` command line administration script you can execute `ejabberdctl commands` (described in the next section, 4.1.1) and also many general `ejabberd commands` (described in section 4.2). This means you can start, stop and perform many other administrative tasks in a local or remote `ejabberd` server (by providing the argument `--node NODENAME`).

The `ejabberdctl` script can be configured in the file `ejabberdctl.cfg`. This file includes detailed information about each configurable option. See section 4.1.2.

The `ejabberdctl` script returns a numerical status code. Success is represented by `0`, error is represented by `1`, and other codes may be used for specific results. This can be used by other scripts to determine automatically if a command succeeded or failed, for example using: `echo $?`

4.1.1 ejabberdctl Commands

When `ejabberdctl` is executed without any parameter, it displays the available options. If there isn't an `ejabberd` server running, the available parameters are:

`start` Start `ejabberd` in background mode. This is the default method.

`debug` Attach an Erlang shell to an already existing `ejabberd` server. This allows to execute commands interactively in the `ejabberd` server.

`live` Start `ejabberd` in live mode: the shell keeps attached to the started server, showing log messages and allowing to execute interactive commands.

If there is an `ejabberd` server running in the system, `ejabberdctl` shows the `ejabberdctl commands` described bellow and all the `ejabberd commands` available in that server (see 4.2.1).

The `ejabberdctl commands` are:

`help` Get help about ejabberdctl or any available command. Try `ejabberdctl help help`.

`status` Check the status of the `ejabberd` server.

`stop` Stop the `ejabberd` server.

`restart` Restart the `ejabberd` server.

`mnesia` Get information about the Mnesia database.

The `ejabberdctl` script can be restricted to require authentication and execute some `ejabberd commands`; see 4.2.2. Add the option to the file `ejabberd.cfg`. In this example there is no restriction:

```
{ejabberdctl_access_commands, []}.
```

If account `robot1@example.org` is registered in `ejabberd` with password `abcdef` (which MD5 is E8B501798950FC58AAD83C8C14978E), and `ejabberd.cfg` contains this setting:

```
{hosts, ["example.org"]}.
{acl, bots, {user, "robot1", "example.org"}}.
{access, ctlaccess, [{allow, bots}]}.
{ejabberdctl_access_commands, [ {ctlaccess, [registered_users, register], []} ]}.
```

then you can do this in the shell:

```
$ ejabberdctl registered_users example.org
Error: no_auth_provided
$ ejabberdctl --auth robot1 example.org E8B501798950FC58AAD83C8C14978E registered_users exampl
robot1
testuser1
testuser2
```

4.1.2 Erlang Runtime System

`ejabberd` is an Erlang/OTP application that runs inside an Erlang runtime system. This system is configured using environment variables and command line parameters. The `ejabberdctl` administration script uses many of those possibilities. You can configure some of them with the file `ejabberdctl.cfg`, which includes detailed description about them. This section describes for reference purposes all the environment variables and command line parameters.

The environment variables:

`EJABBERD_CONFIG_PATH` Path to the ejabberd configuration file.

`EJABBERD_MSGS_PATH` Path to the directory with translated strings.

`EJABBERD_LOG_PATH` Path to the ejabberd service log file.

`EJABBERD_SO_PATH` Path to the directory with binary system libraries.

`EJABBERD_DOC_PATH` Path to the directory with ejabberd documentation.

`EJABBERD_PID_PATH` Path to the PID file that ejabberd can create when started.

`HOME` Path to the directory that is considered `ejabberd`'s home. This path is used to read the file `.erlang.cookie`.

`ERL_CRASH_DUMP` Path to the file where crash reports will be dumped.

`ERL_INETRC` Indicates which IP name resolution to use. If using `-sname`, specify either this option or `-kernel inetrc filepath`.

`ERL_MAX_PORTS` Maximum number of simultaneously open Erlang ports.

`ERL_MAX_ETS_TABLES` Maximum number of ETS and Mnesia tables.

The command line parameters:

`-sname ejabberd` The Erlang node will be identified using only the first part of the host name, i. e. other Erlang nodes outside this domain cannot contact this node. This is the preferable option in most cases.

`-name ejabberd` The Erlang node will be fully identified. This is only useful if you plan to setup an `ejabberd` cluster with nodes in different networks.

`-kernel inetrc '"/etc/ejabberd/inetrc"'` Indicates which IP name resolution to use. If using `-sname`, specify either this option or `ERL_INETRC`.

`-kernel inet_dist_listen_min 4200 inet_dist_listen_min 4210` Define the first and last ports that `epmd` (section 5.2) can listen to.

`-detached` Starts the Erlang system detached from the system console. Useful for running daemons and backgrounds processes.

`-noinput` Ensures that the Erlang system never tries to read any input. Useful for running daemons and backgrounds processes.

`-pa /var/lib/ejabberd/ebin` Specify the directory where Erlang binary files (*.beam) are located.

`-s ejabberd` Tell Erlang runtime system to start the `ejabberd` application.

`-mnesia dir '"/var/lib/ejabberd/"'` Specify the Mnesia database directory.

`-sasl sasl_error_logger {file, "/var/log/ejabberd/erlang.log"}` Path to the Erlang/OTP system log file. SASL here means "System Architecture Support Libraries" not "Simple Authentication and Security Layer".

`+K [true|false]` Kernel polling.

`-smp [auto|enable|disable]` SMP support.

`+P 250000` Maximum number of Erlang processes.

`-remsh ejabberd@localhost` Open an Erlang shell in a remote Erlang node.

`-hidden` The connections to other nodes are hidden (not published). The result is that this node is not considered part of the cluster. This is important when starting a temporary `ctl` or `debug` node.

Note that some characters need to be escaped when used in shell scripts, for instance `"` and `{}`. You can find other options in the Erlang manual page (`erl -man erl`).

4.2 `ejabberd` Commands

An `ejabberd command` is an abstract function identified by a name, with a defined number and type of calling arguments and type of result that is registered in the `ejabberd_commands` service. Those commands can be defined in any Erlang module and executed using any valid frontend.

`ejabberd` includes a frontend to execute `ejabberd commands`: the script `ejabberdctl`. Other known frontends that can be installed to execute ejabberd commands in different ways are: `ejabberd_xmlrpc` (XML-RPC service), `mod_rest` (HTTP POST service), `mod_shcommands` (ejabberd WebAdmin page).

4.2.1 List of ejabberd Commands

`ejabberd` includes a few ejabberd Commands by default. When more modules are installed, new commands may be available in the frontends.

The easiest way to get a list of the available commands, and get help for them is to use the ejabberdctl script:

```
$ ejabberdctl help
Usage: ejabberdctl [--node nodename] [--auth user host password] command [options]

Available commands in this ejabberd node:
  backup file                  Store the database to backup file
  connected_users              List all established sessions
  connected_users_number       Get the number of established sessions
  ...
```

The most interesting ones are:

`reopen_log` Reopen the log files after they were renamed. If the old files were not renamed before calling this command, they are automatically renamed to `"*-old.log"`. See section 7.1.

`backup ejabberd.backup` Store internal Mnesia database to a binary backup file.

`restore ejabberd.backup` Restore immediately from a binary backup file the internal Mnesia database. This will consume a lot of memory if you have a large database, so better use `install_fallback`.

`install_fallback ejabberd.backup` The binary backup file is installed as fallback: it will be used to restore the database at the next ejabberd start. This means that, after running this command, you have to restart ejabberd. This command requires less memory than `restore`.

`dump ejabberd.dump` Dump internal Mnesia database to a text file dump.

`load ejabberd.dump` Restore immediately from a text file dump. This is not recommended for big databases, as it will consume much time, memory and processor. In that case it's preferable to use `backup` and `install_fallback`.

`import_piefxis, export_piefxis, export_piefxis_host` These options can be used to migrate accounts using XEP-0227[1] formatted XML files from/to other Jabber/XMPP servers or move users of a vhost to another ejabberd installation. See also ejabberd migration kit[2].

`import_file, import_dir` These options can be used to migrate accounts using jabberd1.4 formatted XML files. from other Jabber/XMPP servers There exist tutorials to migrate from other software to ejabberd[3].

`delete_expired_messages` This option can be used to delete old messages in offline storage. This might be useful when the number of offline messages is very high.

`delete_old_messages days` Delete offline messages older than the given days.

`register user host password` Register an account in that domain with the given password.

`unregister user host` Unregister the given account.

4.2.2 Restrict Execution with AccessCommands

The frontends can be configured to restrict access to certain commands. In that case, authentication information must be provided. In each frontend the `AccessCommands` option is defined in a different place. But in all cases the option syntax is the same:

```
AccessCommands = [ {Access, CommandNames, Arguments}, ...]
Access = atom()
CommandNames = all | [CommandName]
CommandName = atom()
Arguments = [ {ArgumentName, ArgumentValue}, ...]
ArgumentName = atom()
ArgumentValue = any()
```

[1] http://xmpp.org/extensions/xep-0227.html
[2] https://support.process-one.net/doc/display/MESSENGER/ejabberd+migration+kit
[3] http://www.ejabberd.im/migrate-to-ejabberd

The default value is to not define any restriction: `[]`. The authentication information is provided when executing a command, and is Username, Hostname and Password of a local XMPP account that has permission to execute the corresponding command. This means that the account must be registered in the local ejabberd, because the information will be verified. It is possible to provide the plaintext password or its MD5 sum.

When one or several access restrictions are defined and the authentication information is provided, each restriction is verified until one matches completely: the account matches the Access rule, the command name is listed in CommandNames, and the provided arguments do not contradict Arguments.

As an example to understand the syntax, let's suppose those options:

```
{hosts, ["example.org"]}.
{acl, bots, {user, "robot1", "example.org"}}.
{access, commaccess, [{allow, bots}]}.
```

This list of access restrictions allows only `robot1@example.org` to execute all commands:

```
[{commaccess, all, []}]
```

See another list of restrictions (the corresponding ACL and ACCESS are not shown):

```
[
 %% This bot can execute all commands:
 {bot, all, []},
 %% This bot can only execute the command 'dump'. No argument restriction:
 {bot_backups, [dump], []}
 %% This bot can execute all commands,
 %% but if a 'host' argument is provided, it must be "example.org":
 {bot_all_example, all, [{host, "example.org"}]},
 %% This bot can only execute the command 'register',
 %% and if argument 'host' is provided, it must be "example.org":
 {bot_reg_example, [register], [{host, "example.org"}]},
 %% This bot can execute the commands 'register' and 'unregister',
 %% if argument host is provided, it must be "test.org":
 {_bot_reg_test, [register, unregister], [{host, "test.org"}]}
]
```

4.3 Web Admin

The `ejabberd` Web Admin allows to administer most of `ejabberd` using a web browser.

This feature is enabled by default: a `ejabberd_http` listener with the option `web_admin` (see section 3.1.3) is included in the listening ports. Then you can open `http://server:port/admin/` in your favourite web browser. You will be asked to enter the username (the *full* Jabber ID)

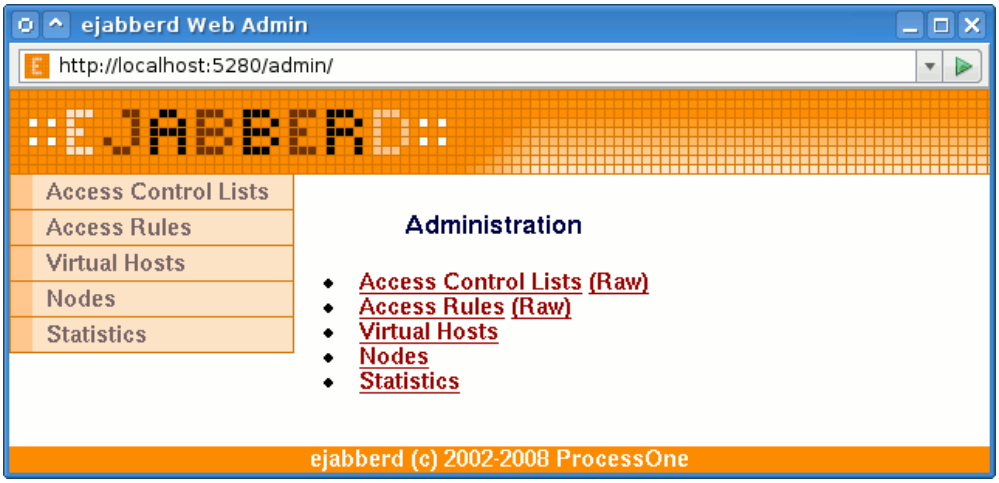

Figure 4.1: Top page from the Web Admin

and password of an `ejabberd` user with administrator rights. After authentication you will see a page similar to figure 4.1.

Here you can edit access restrictions, manage users, create backups, manage the database, enable/disable ports listened for, view server statistics,...

The access rule `configure` determines what accounts can access the Web Admin and modify it. The access rule `webadmin_view` is to grant only view access: those accounts can browse the Web Admin with read-only access.

Example configurations:

- You can serve the Web Admin on the same port as the HTTP Polling interface. In this example you should point your web browser to `http://example.org:5280/admin/` to administer all virtual hosts or to `http://example.org:5280/admin/server/example.com/` to administer only the virtual host `example.com`. Before you get access to the Web Admin you need to enter as username, the JID and password from a registered user that is allowed to configure `ejabberd`. In this example you can enter as username '`admin@example.net`' to administer all virtual hosts (first URL). If you log in with '`admin@example.com`' on `http://example.org:5280/admin/server/example.com/` you can only administer the virtual host `example.com`. The account '`reviewer@example.com`' can browse that vhost in read-only mode.

```
{acl, admins, {user, "admin", "example.net"}}.
{host_config, "example.com", [{acl, admins, {user, "admin", "example.com"}}]}.
{host_config, "example.com", [{acl, viewers, {user, "reviewer", "example.com"}}]}.

{access, configure, [{allow, admins}]}.
{access, webadmin_view, [{allow, viewers}]}.

{hosts, ["example.org"]}.
```

```
{listen,
 [
  ...
  {5280, ejabberd_http, [http_poll, web_admin]},
  ...
 ]}.
```

- For security reasons, you can serve the Web Admin on a secured connection, on a port differing from the HTTP Polling interface, and bind it to the internal LAN IP. The Web Admin will be accessible by pointing your web browser to `https://192.168.1.1:5282/admin/`:

```
{hosts, ["example.org"]}.

{listen,
 [
  ...
  {5280, ejabberd_http, [
                         http_poll
                        ]},
  {{5282, "192.168.1.1"}, ejabberd_http, [
                                          web_admin,
                                          tls, {certfile, "/usr/local/etc/server.pem"}
                                         ]},
  ...
 ]}.
```

Certain pages in the ejabberd Web Admin contain a link to a related section in the ejabberd Installation and Operation Guide. In order to view such links, a copy in HTML format of the Guide must be installed in the system. The file is searched by default in `"/share/doc/ejabberd/guide.html"`. The directory of the documentation can be specified in the environment variable `EJABBERD_DOC_PATH`. See section 4.1.2.

4.4 Ad-hoc Commands

If you enable `mod_configure` and `mod_adhoc`, you can perform several administrative tasks in `ejabberd` with a XMPP client. The client must support Ad-Hoc Commands (XEP-0050[4]), and you must login in the XMPP server with an account with proper privileges.

4.5 Change Computer Hostname

`ejabberd` uses the distributed Mnesia database. Being distributed, Mnesia enforces consistency of its file, so it stores the name of the Erlang node in it (see section 5.4). The name of an Erlang

[4] http://xmpp.org/extensions/xep-0050.html

node includes the hostname of the computer. So, the name of the Erlang node changes if you change the name of the machine in which `ejabberd` runs, or when you move `ejabberd` to a different machine.

You have two ways to use the old Mnesia database in an ejabberd with new node name: put the old node name in `ejabberdctl.cfg`, or convert the database to the new node name.

Those example steps will backup, convert and load the Mnesia database. You need to have either the old Mnesia spool dir or a backup of Mnesia. If you already have a backup file of the old database, you can go directly to step 5. You also need to know the old node name and the new node name. If you don't know them, look for them by executing `ejabberdctl` or in the ejabberd log files.

Before starting, setup some variables:

```
OLDNODE=ejabberd@oldmachine
NEWNODE=ejabberd@newmachine
OLDFILE=/tmp/old.backup
NEWFILE=/tmp/new.backup
```

1. Start ejabberd enforcing the old node name:

   ```
   ejabberdctl --node $OLDNODE start
   ```

2. Generate a backup file:

   ```
   ejabberdctl --node $OLDNODE backup $OLDFILE
   ```

3. Stop the old node:

   ```
   ejabberdctl --node $OLDNODE stop
   ```

4. Make sure there aren't files in the Mnesia spool dir. For example:

   ```
   mkdir /var/lib/ejabberd/oldfiles
   mv /var/lib/ejabberd/*.* /var/lib/ejabberd/oldfiles/
   ```

5. Start ejabberd. There isn't any need to specify the node name anymore:

   ```
   ejabberdctl start
   ```

6. Convert the backup to new node name:

   ```
   ejabberdctl mnesia_change_nodename $OLDNODE $NEWNODE $OLDFILE $NEWFILE
   ```

7. Install the backup file as a fallback:

   ```
   ejabberdctl install_fallback $NEWFILE
   ```

8. Stop ejabberd:

```
ejabberdctl stop
```

You may see an error message in the log files, it's normal, so don't worry:

```
Mnesia(ejabberd@newmachine):
** ERROR ** (ignoring core)
** FATAL ** A fallback is installed and Mnesia must be restarted.
  Forcing shutdown after mnesia_down from ejabberd@newmachine...
```

9. Now you can finally start ejabberd:

```
ejabberdctl start
```

10. Check that the information of the old database is available: accounts, rosters... After you finish, remember to delete the temporary backup files from public directories.

Chapter 5

Securing ejabberd

5.1 Firewall Settings

You need to take the following TCP ports in mind when configuring your firewall:

Port	Description
5222	Standard port for Jabber/XMPP client connections, plain or STARTTLS.
5223	Standard port for Jabber client connections using the old SSL method.
5269	Standard port for Jabber/XMPP server connections.
4369	EPMD (section 5.2) listens for Erlang node name requests.
port range	Used for connections between Erlang nodes. This range is configurable (see section 5.2).

5.2 epmd

epmd (Erlang Port Mapper Daemon)[1] is a small name server included in Erlang/OTP and used by Erlang programs when establishing distributed Erlang communications. `ejabberd` needs `epmd` to use `ejabberdctl` and also when clustering `ejabberd` nodes. This small program is automatically started by Erlang, and is never stopped. If `ejabberd` is stopped, and there aren't any other Erlang programs running in the system, you can safely stop `epmd` if you want.

`ejabberd` runs inside an Erlang node. To communicate with `ejabberd`, the script `ejabberdctl` starts a new Erlang node and connects to the Erlang node that holds `ejabberd`. In order for this communication to work, `epmd` must be running and listening for name requests in the port 4369. You should block the port 4369 in the firewall in such a way that only the programs in your machine can access it.

If you build a cluster of several `ejabberd` instances, each `ejabberd` instance is called an `ejabberd` node. Those `ejabberd` nodes use a special Erlang communication method to build the cluster,

[1] http://www.erlang.org/doc/man/epmd.html

and EPMD is again needed listening in the port 4369. So, if you plan to build a cluster of `ejabberd` nodes you must open the port 4369 for the machines involved in the cluster. Remember to block the port so Internet doesn't have access to it.

Once an Erlang node solved the node name of another Erlang node using EPMD and port 4369, the nodes communicate directly. The ports used in this case by default are random, but can be configured in the file `ejabberdctl.cfg`. The Erlang command-line parameter used internally is, for example:

```
erl ... -kernel inet_dist_listen_min 4370 inet_dist_listen_max 4375
```

5.3 Erlang Cookie

The Erlang cookie is a string with numbers and letters. An Erlang node reads the cookie at startup from the command-line parameter `-setcookie`. If not indicated, the cookie is read from the cookie file `$HOME/.erlang.cookie`. If this file does not exist, it is created immediately with a random cookie. Two Erlang nodes communicate only if they have the same cookie. Setting a cookie on the Erlang node allows you to structure your Erlang network and define which nodes are allowed to connect to which.

Thanks to Erlang cookies, you can prevent access to the Erlang node by mistake, for example when there are several Erlang nodes running different programs in the same machine.

Setting a secret cookie is a simple method to difficult unauthorized access to your Erlang node. However, the cookie system is not ultimately effective to prevent unauthorized access or intrusion to an Erlang node. The communication between Erlang nodes are not encrypted, so the cookie could be read sniffing the traffic on the network. The recommended way to secure the Erlang node is to block the port 4369.

5.4 Erlang Node Name

An Erlang node may have a node name. The name can be short (if indicated with the command-line parameter `-sname`) or long (if indicated with the parameter `-name`). Starting an Erlang node with -sname limits the communication between Erlang nodes to the LAN.

Using the option `-sname` instead of `-name` is a simple method to difficult unauthorized access to your Erlang node. However, it is not ultimately effective to prevent access to the Erlang node, because it may be possible to fake the fact that you are on another network using a modified version of Erlang `epmd`. The recommended way to secure the Erlang node is to block the port 4369.

5.5 Securing Sensitive Files

`ejabberd` stores sensitive data in the file system either in plain text or binary files. The file system permissions should be set to only allow the proper user to read, write and execute those

files and directories.

`ejabberd configuration file: /etc/ejabberd/ejabberd.cfg` Contains the JID of administrators and passwords of external components. The backup files probably contain also this information, so it is preferable to secure the whole `/etc/ejabberd/` directory.

`ejabberd service log: /var/log/ejabberd/ejabberd.log` Contains IP addresses of clients. If the loglevel is set to 5, it contains whole conversations and passwords. If a logrotate system is used, there may be several log files with similar information, so it is preferable to secure the whole `/var/log/ejabberd/` directory.

`Mnesia database spool files in /var/lib/ejabberd/` The files store binary data, but some parts are still readable. The files are generated by Mnesia and their permissions cannot be set directly, so it is preferable to secure the whole `/var/lib/ejabberd/` directory.

`Erlang cookie file: /var/lib/ejabberd/.erlang.cookie` See section 5.3.

Chapter 6

Clustering

6.1 How it Works

A XMPP domain is served by one or more `ejabberd` nodes. These nodes can be run on different machines that are connected via a network. They all must have the ability to connect to port 4369 of all another nodes, and must have the same magic cookie (see Erlang/OTP documentation, in other words the file `~ejabberd/.erlang.cookie` must be the same on all nodes). This is needed because all nodes exchange information about connected users, s2s connections, registered services, etc...

Each `ejabberd` node has the following modules:

- router,
- local router,
- session manager,
- s2s manager.

6.1.1 Router

This module is the main router of XMPP packets on each node. It routes them based on their destination's domains. It uses a global routing table. The domain of the packet's destination is searched in the routing table, and if it is found, the packet is routed to the appropriate process. If not, it is sent to the s2s manager.

6.1.2 Local Router

This module routes packets which have a destination domain equal to one of this server's host names. If the destination JID has a non-empty user part, it is routed to the session manager, otherwise it is processed depending on its content.

6.1.3 Session Manager

This module routes packets to local users. It looks up to which user resource a packet must be sent via a presence table. Then the packet is either routed to the appropriate c2s process, or stored in offline storage, or bounced back.

6.1.4 s2s Manager

This module routes packets to other XMPP servers. First, it checks if an opened s2s connection from the domain of the packet's source to the domain of the packet's destination exists. If that is the case, the s2s manager routes the packet to the process serving this connection, otherwise a new connection is opened.

6.2 Clustering Setup

Suppose you already configured `ejabberd` on one machine named (`first`), and you need to setup another one to make an `ejabberd` cluster. Then do following steps:

1. Copy `~ejabberd/.erlang.cookie` file from `first` to `second`.

   (alt) You can also add '`-setcookie content_of_.erlang.cookie`' option to all '`erl`' commands below.

2. On `second` run the following command as the `ejabberd` daemon user, in the working directory of `ejabberd`:

   ```
   erl -sname ejabberd \
       -mnesia dir '"/var/lib/ejabberd/"' \
       -mnesia extra_db_nodes "['ejabberd@first']" \
       -s mnesia
   ```

   This will start Mnesia serving the same database as `ejabberd@first`. You can check this by running the command '`mnesia:info().`'. You should see a lot of remote tables and a line like the following:

   Note: the Mnesia directory may be different in your system. To know where does ejabberd expect Mnesia to be installed by default, call 4.1 without options and it will show some help, including the Mnesia database spool dir.

   ```
   running db nodes   = [ejabberd@first, ejabberd@second]
   ```

3. Now run the following in the same '`erl`' session:

   ```
   mnesia:change_table_copy_type(schema, node(), disc_copies).
   ```

This will create local disc storage for the database.

(alt) Change storage type of the `scheme` table to 'RAM and disc copy' on the second node via the Web Admin.

4. Now you can add replicas of various tables to this node with '`mnesia:add_table_copy`' or '`mnesia:change_table_copy_type`' as above (just replace '`schema`' with another table name and '`disc_copies`' can be replaced with '`ram_copies`' or '`disc_only_copies`').

   Which tables to replicate is very dependant on your needs, you can get some hints from the command '`mnesia:info().`', by looking at the size of tables and the default storage type for each table on 'first'.

   Replicating a table makes lookups in this table faster on this node. Writing, on the other hand, will be slower. And of course if machine with one of the replicas is down, other replicas will be used.

   Also section 5.3 (Table Fragmentation) of Mnesia User's Guide[1] can be helpful.

   (alt) Same as in previous item, but for other tables.

5. Run '`init:stop().`' or just '`q().`' to exit from the Erlang shell. This probably can take some time if Mnesia has not yet transfered and processed all data it needed from `first`.

6. Now run `ejabberd` on `second` with a configuration similar as on `first`: you probably do not need to duplicate '`acl`' and '`access`' options because they will be taken from `first`; and `mod_irc` should be enabled only on one machine in the cluster.

You can repeat these steps for other machines supposed to serve this domain.

6.3 Service Load-Balancing

6.3.1 Components Load-Balancing

6.3.2 Domain Load-Balancing Algorithm

`ejabberd` includes an algorithm to load balance the components that are plugged on an `ejabberd` cluster. It means that you can plug one or several instances of the same component on each `ejabberd` cluster and that the traffic will be automatically distributed.

The default distribution algorithm try to deliver to a local instance of a component. If several local instances are available, one instance is chosen randomly. If no instance is available locally, one instance is chosen randomly among the remote component instances.

If you need a different behaviour, you can change the load balancing behaviour with the option `domain_balancing`. The syntax of the option is the following:

```
{domain_balancing, "component.example.com", BalancingCriteria}.
```

[1] http://www.erlang.org/doc/apps/mnesia/Mnesia_chap5.html#5.3

Several balancing criteria are available:

- `destination`: the full JID of the packet `to` attribute is used.
- `source`: the full JID of the packet `from` attribute is used.
- `bare_destination`: the bare JID (without resource) of the packet `to` attribute is used.
- `bare_source`: the bare JID (without resource) of the packet `from` attribute is used.

If the value corresponding to the criteria is the same, the same component instance in the cluster will be used.

6.3.3 Load-Balancing Buckets

When there is a risk of failure for a given component, domain balancing can cause service trouble. If one component is failing the service will not work correctly unless the sessions are rebalanced.

In this case, it is best to limit the problem to the sessions handled by the failing component. This is what the `domain_balancing_component_number` option does, making the load balancing algorithm not dynamic, but sticky on a fix number of component instances.

The syntax is:

```
{domain_balancing_component_number, "component.example.com", Number}.
```

Chapter 7

Debugging

7.1 Log Files

An `ejabberd` node writes two log files:

`ejabberd.log` is the ejabberd service log, with the messages reported by `ejabberd` code

`erlang.log` is the Erlang/OTP system log, with the messages reported by Erlang/OTP using SASL (System Architecture Support Libraries)

The option `loglevel` modifies the verbosity of the file ejabberd.log. The syntax is one of:

`{loglevel, Level}`. The standard form to set a global log level.

`{loglevel, {Level, [{Module, Level}, ...]}}`. The given Erlang modules will be logged with specific log levels, all others will use the default log level.

The possible `Level` are:

`0` No ejabberd log at all (not recommended)

`1` Critical

`2` Error

`3` Warning

`4` Info

`5` Debug

For example, the default configuration is:

```
{loglevel, 4}.
```

The log files grow continually, so it is recommended to rotate them periodically. To rotate the log files, rename the files and then reopen them. The ejabberdctl command `reopen-log` (please refer to section 4.1.1) reopens the log files, and also renames the old ones if you didn't rename them.

7.2 Debug Console

The Debug Console is an Erlang shell attached to an already running `ejabberd` server. With this Erlang shell, an experienced administrator can perform complex tasks.

This shell gives complete control over the `ejabberd` server, so it is important to use it with extremely care. There are some simple and safe examples in the article Interconnecting Erlang Nodes[1]

To exit the shell, close the window or press the keys: control+c control+c.

7.3 Watchdog Alerts

`ejabberd` includes a watchdog mechanism that may be useful to developers when troubleshooting a problem related to memory usage. If a process in the `ejabberd` server consumes more memory than the configured threshold, a message is sent to the XMPP accounts defined with the option `watchdog_admins` in the `ejabberd` configuration file.

The syntax is:

```
{watchdog_admins, [JID, ...]}.
```

The memory consumed is measured in `words`: a word on 32-bit architecture is 4 bytes, and a word on 64-bit architecture is 8 bytes. The threshold by default is 1000000 words. This value can be configured with the option `watchdog_large_heap`, or in a conversation with the watchdog alert bot.

The syntax is:

```
{watchdog_large_heap, Number}.
```

Example configuration:

```
{watchdog_admins, ["admin2@localhost", "admin2@example.org"]}.
{watchdog_large_heap, 30000000}.
```

[1] http://www.ejabberd.im/interconnect-erl-nodes

To remove watchdog admins, remove them in the option. To remove all watchdog admins, set the option with an empty list:

```
{watchdog_admins, []}.
```

Appendix A

Internationalization and Localization

The source code of `ejabberd` supports localization. The translators can edit the gettext[1] .po files using any capable program (KBabel, Lokalize, Poedit...) or a simple text editor.

Then gettext is used to extract, update and export those .po files to the .msg format read by `ejabberd`. To perform those management tasks, in the `src/` directory execute `make translations`. The translatable strings are extracted from source code to generate the file `ejabberd.pot`. This file is merged with each .po file to produce updated .po files. Finally those .po files are exported to .msg files, that have a format easily readable by `ejabberd`.

All built-in modules support the `xml:lang` attribute inside IQ queries. Figure A.1, for example, shows the reply to the following query:

```
<iq id='5'
    to='example.org'
    type='get'
    xml:lang='ru'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

The Web Admin also supports the `Accept-Language` HTTP header.

[1] http://www.gnu.org/software/gettext/

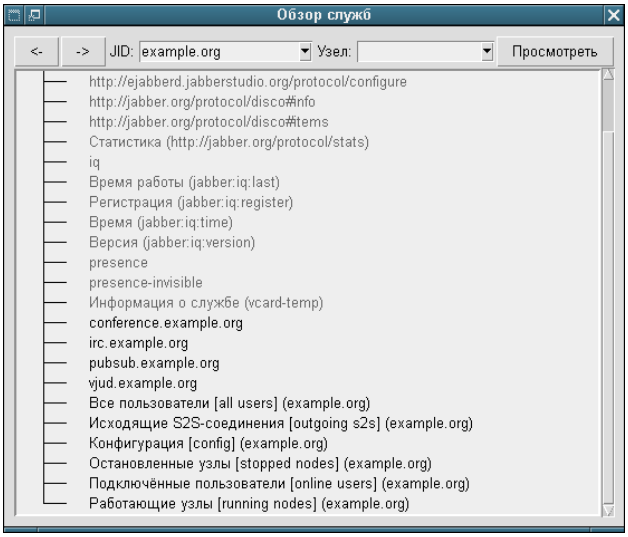

Figure A.1: Service Discovery when `xml:lang='ru'`

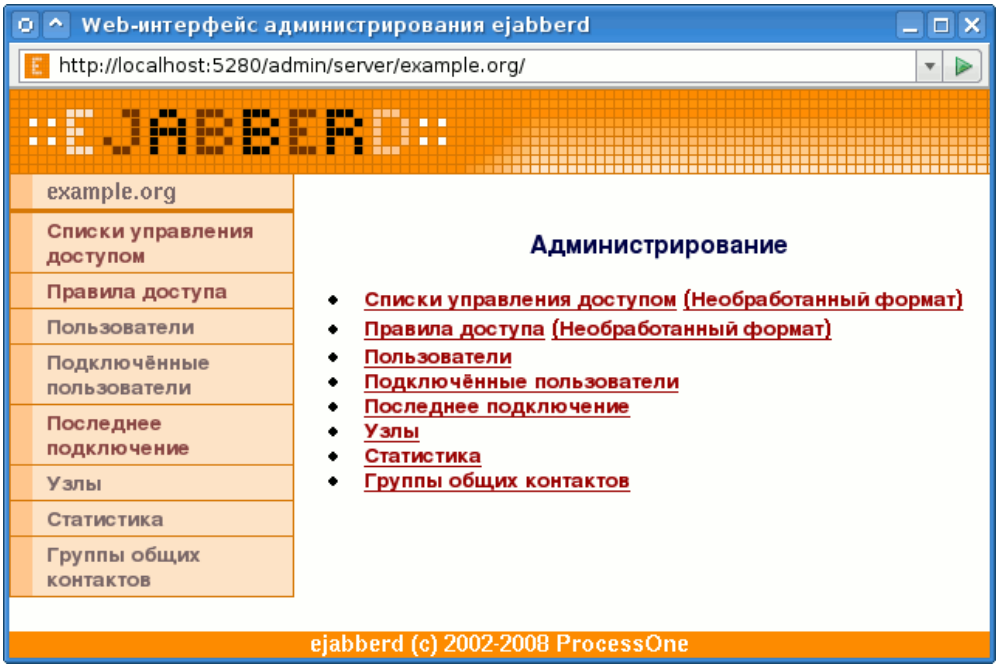

Figure A.2: Web Admin showing a virtual host when the web browser provides the HTTP header 'Accept-Language: ru'

Appendix B

Release Notes

Release notes are available from ejabberd Home Page[1]

[1] http://www.process-one.net/en/ejabberd/release_notes/

Appendix C

Acknowledgements

Thanks to all people who contributed to this guide:

- Alexey Shchepin (xmpp:aleksey@jabber.ru)
- Badlop (xmpp:badlop@jabberes.org)
- Evgeniy Khramtsov (xmpp:xram@jabber.ru)
- Florian Zumbiehl (xmpp:florz@florz.de)
- Marcin Owsiany (xmpp:marcin.owsiany@gmail.com)
- Michael Grigutsch (xmpp:migri@jabber.i-pobox.net)
- Mickael Remond (xmpp:mremond@process-one.net)
- Sander Devrieze (xmpp:s.devrieze@gmail.com)
- Sergei Golovan (xmpp:sgolovan@nes.ru)
- Vsevolod Pelipas (xmpp:vsevoload@jabber.ru)

Appendix D

Copyright Information

Index