Jabberd 2 Installation and Administration Guide

Table of Contents

Appendices (including Quickstart Guide and FAQ):

1. Introduction

This document comprises the installation and administrative guide for the Jabberd 2 Server (Jabberd), the latest release of the popular open source messaging system based on the Jabber Protocol. The goal of Jabber is to provide an XML protocol for synchronous and asynchronous communication for client to client, client to server, and server to server messaging, although the primary use for Jabber is instant messaging (IM).

The Jabberd server is the original open−source server implementation of the Jabber protocol, and it remains the most popular software for deploying Jabber either inside a company or as a public IM service. Jeremie Miller initiated the Jabber Project in 1998 as a free and open alternative to proprietary IM services. The Jabberd server continues to be the core of the Jabber Project, and Jabberd 2 is the successor to the widely used Jabberd 1.4 server. Jabberd 2 is based on a completely new code base with a new architecture, additional features and improved adherence to the Jabber protocol.

The creation of a common messaging protocol, now known as XMPP (Extensible Messaging and Presence Protocol), has allowed for the creation of numerous Jabber server implementations in addition to the Jabberd server. Among these are several open source projects, including WP Jabber and ejabberd, as well as several commercial offerings from companies such as i3connect, Jabcast, Tipic and Jabber, Inc.

1.1. Purpose and Scope

The authors of this document intend to provide a complete guide for jabberd 2 installation, administration and development:

- System Preparation
- Server Installation
- Server Configuration
- Architecture Overview

The intended audience are people who wish to install and/or maintain a jabberd 2 server on Unix, or one of its variants. As such, this document covers jabberd installation on Unix operating systems only.

1.2. Required Background

The authors have made every attempt to make this a step−by−step guide; however, some familiarity with a Unix or Linux operating system is assumed:

- Ability to use a Linux command−prompt console
- Familiarity with the file system on which your server is to be installed
- Familiarity with a text editor, such as vi, Nano or NEdit
- Basic ability to edit XML files

It is assumed that the reader has a basic familiarity with using a Jabber client. Additionally, it is assumed that the reader is familiar with hardware and software, such as a firewall, router or modem, that the jabber server will use to connect to the Internet if any such hardware is used. Configuration of these secondary programs and devices is beyond the scope of this guide.

1.3. How to Use This Document

This guide is organized into sections grouped according to intended use by the user:

- Installing and Configuring Jabberd 2 (Sections 2−4)
- Common Configuration and Administration Tasks (Sections 5−6)
- Overview of Jabberd 2 Configuration Files (Sections 7−12)
- Jabber 2 Architecture (Section 13)
- Jabberd Quickstart Guide (Appendix A)
- Jabberd 2 FAQ

The Quickstart Guide is designed to get experienced users up and running quickly. Detailed Jabberd 2 installation instructions that begin in Section 2, Preparing to Install Jabberd 2. Sections 5 and 6 list common Configuration and Administration tasks, respectively, while the remaining sections provide detailed configuration information.

1.4. Conventions Used in this Document

This document is provided primarily as an installation guide, and as such, special conventions are used to make installation easier for the user. The conventions below appear throughout the installation sections of this guide:

Table 1.4. Document Conventions Used in this Guide

Convention	Name	Description
P	Parameter	Information about your specific setup that you will enter into a configuration file, etc.
C	Checkpoint	A point at which to stop and check your setup.
N	Note	An informational note.
I	Important	An important note or warning.
F	Required Files	Software or specific files needed to complete a step or series of steps.
O	Optional Step	A step that is not required for the most basic installation.
E	External System	A step that may require configuration on an external system, such as a router.

The most useful of these conventions are "Parameters." Section 2 begins with a list of information about your setup (Parameters) that you will need during the installation steps. You can gather all this information before you start installing Jabberd, and then you can refer back to your list for every step that displays a ***P***.

The installation guide is organized into numbered steps and sub−steps, etc. Users are encouraged to use the guide as a check list. When all of the sub−steps of a step are completed, then the parent step itself is also completed. Note that all children (sub−steps) of an "Optional Step" are optional also. Note that the "Optional Step" designation provides information about the conditions and/or requirements under which the optional step should be performed.

Steps labeled with "External System" provide the user with information about set up that may need to be performed on a system external to jabberd. These systems include routers, firewalls, DNS servers, etc. "External System" notes are intended to be informational rather than comprehensive. The remaining conventions are self−explanatory.

Note that each command listed in this guide refers to a command entered at a command prompt or at a command prompt shell, such as X−term or E−term. Note also that in this document, the term "Jabberd" refers to the Jabberd 2 server, except where noted. The term "Jabber" refers to Jabber−based system or systems, and "XMPP" refers to the protocol over which Jabber systems run.

This document was created using Structured Text. Structured Text uses standard text formatting conventions, such as underscores, to represent formatted text. Efforts have been made to keep the text and HTML representations close. One exception is the use of a bang character ("!") in the text version to escape unwanted formatting in the HTML version. "Bangs" appear at the beginning of some lines in the text version, and these can be ignored. They do not appear in the HTML version.

1.5. Further Reading

Visit the Jabber Software Foundation for the latest news about jabberd 2, jabber clients, and the Jabber Protocol. The Jabber Faq answers basic questions about Jabber. Readers are encouraged to visit the Jadmin Archive for questions about jabber administration, or subscribe to the Jadmin Mailing List for the most up to

date jabberd administration information. Jabberd 2 development information can be found in the Jabberd Archive or by subscribing to the Jabberd List.

There are also several good books about Jabber. Note that books below detail *jabberd 1.4 only* as of this writing in 2003:

- Jabberd Administration Guide for version 1.4
- Programming Jabber by D.J. Adams
- Instant Messaging for Java by Iain Shigeoka
- Jabber Programming by Stephen Lee and Terence Smelser
- Jabber Developer's Handbook from Sams Publishing

1.6. Legalese

1.7. Sources

Robert Norris, who is the primary Jabberd 2 developer, and the posters on the Jabberd List provided technical information for this document.

Back	Up	Next

2. Preparation for Jabberd 2

This section will prepare you and your system to install Jabberd 2:

- Gather Required Information
- Create Jabber User and Group
- Create Directories for Logging and PID's
- Install Prerequisites

The table in Section 2.1. lists information required for Jabberd 2 installation. Collecting this information at this point is optional; however, completing this table now will make installation easier.

2.1. Gather Required Information

The table below lists information that will be required during the installation process. Information is provided for each parameter below:

- Parameter
- Required
- Section
- Description

- Your Information

"Parameter" is the name of the piece of information. Throughout this guide, specific parameters are referenced with the ***P*** convention. "Required is either "Y" (yes), "N" (no), or *Option Name*. *Option Name* refers to the option for which the parameter is necessary. Some entries have suggested entries for "Your Information." These entries represent either limited choices or default values. Where default values are given, they are used in the examples in this guide.

For a minimal installation, complete all the required parameters. For more information about the conditions under which an optional parameter is required, see the referenced section. "Description" is a short description. Again, see the referenced section for more detailed information.

I *Important: Table Contains Passwords*

The table below contains passwords. If you write your passwords in the table below, store this document (or page) in a secure location.

Table 2.1. Required Information for Jabberd 2 Installation

Parameter	Required	Section(s)	Description	Your Information
Jabberd User and Group	Y	2.2	The Linux (or other OS) user and group that will be used to run Jabberd	*user:* jabber *group:* jabber
PID Directory	Y	2.3	Directory in which Jabberd stores PID Files	/usr/local/var/jabberd/pid
Log Directory	N	2.3	Directory for Jabberd logs. If not specified in configuration files, logging defaults to syslog.	/usr/local/var/jabberd/log
Authentication Package	Y	2.4.3 3.3,	Third party package to be used for Jabberd authentication management	MySQL, PostgreSQL, Berkeley DB, OpenLDAP or PAM
Data Storage Package	Y	2.4.4 4.3	Third party package to be used for storage of Jabberd data	MySQL, PostgreSQL or Berkeley DB
Data Directory	Berkeley DB	4.1.1, 4.2.1,	Directory for Berkeley DB data files	/usr/local/var/jabberd/db
MySQL User and Password	MySQL	3.5.2.2, 4.1.2, 4.2.2,	MySQL user and password that Jabberd uses to connect to MySQL	*user:* jabberd2 *password:* secret
PostgreSQL User and Password	PostgreSQL	3.5.3.1, 1 3, 4.2.3,	PostgreSQL user and password that Jabberd uses to connect to PostgreSQL	*user:* jabberd2 *password:* secret
OpenLDAP Connection Settings	OpenLDAP	2 5	Connection settings for your OpenLDAP server: FQDN for LDAP server (or IP), port, and LDAP version used (either v2 or v3)	

OpenLDAP User and Password	OpenLDAP	4.2.5	User and password needed to connect to your OpenLDAP server. Required only if your OpenLDAP server does not permit anonymous binding (access)	
OpenLDAP Query Settings	OpenLDAP	4.2.5	Base DN (distinguished name) and User ID attribute used to build queries for the OpenLDAP server. Base DN can be either the server root DN or an RDN (relative distinguished name) under which User ID's are found.	
Hostname	Y	4.4	Hostname on which your Jabberd server is to be installed. For Internet accessible servers, this would be something like `somedomain.com`	
SSL Key Location	N	5.3.1, 5.3.2	Location of OpenSSL pemfile. Required for SSL−encrypted communication	/usr/local/etc/jabberd/server.pem
Router User and Password	N	5.4	User and password used for component connections with the Jabberd Router component	*user:* jabberd2 *password:* secret

2.2. Create Jabber User and Group

You should create a `jabber` user and group to run the server:

P *Parameter: Jabber User and Group*

Create a user and group that will be used to run Jabberd (as superuser):

```
su
groupadd jabber
useradd −g jabber jabber
```

I *Important: Check Your User and Group Commands*

The above commands are intended as an example. The commands and parameters for adding a user and group may vary for your system. Consult your manuals if you have any doubt about these commands.

2.3. Create Directories for PID's and Logs

You should create a directory for Jabberd to store its PID and log files, and ownership of these directories should be set to the user created above.

P *Parameter: PID Directory*

Create a directory for PID files (as superuser):

```
su
mkdir -p /usr/local/var/jabberd/pid/
chown -R jabber:jabber /usr/local/var/jabberd/pid/
```

The above directory is the default location for Jabberd PID files.

You may also choose to create a directory for Jabberd logs.

O *Optional: Log Directory*

If you wish, create a separate directory for Jabberd logs, and set ownership to your jabber user:

```
mkdir -p /usr/local/var/jabberd/log/
chown -R jabber:jabber /usr/local/var/jabberd/log
```

N *Note: Log Files Default to Syslog*

Note that Jabberd writes messages to `syslog` by default. In order to force Jabberd to write its logs to the directory above, the component XML files must be edited to specifiy the directory created above.

2.4. Install Prerequisites

Jabberd 2 has four prerequisites:

- OpenSSL (version 0.9.6b or higher)
- Libidn (version 0.3.0 or higher)
- Data Storage Package
- Authentication Package

Technically, Jabberd 2 can be installed without OpenSSL or Libidn; however, it is *strongly* recommended that these packages be installed prior to installing Jabberd. Jabberd 2 also requires data packages for application and authentication; however, a single package, such as MySQL, can be used to satisfy requirements for both data storage and authentication.

2.4.1. OpenSSL

OpenSSL provides encrypted client to server and server to server communication for Jabber. The XMPP Protocol requires that Jabber servers support TLS (Transport Security Layer). TLS is the successor to SSL.

N *Note: Minimum OpenSSL Version*

Jabberd 2 relies on OpenSSL versions 0.9.6b or higher.

I *Important: OpenSSL Upgrade Issues*

If you upgrade OpenSSL, you may need to recompile installed software that currently relies on an older version of OpenSSL. This warning is provided because many utilities rely on OpenSSL, and these may cease to function after OpenSSL is upgraded. Caution is recommended when upgrading OpenSSL, and detailed instructions for upgrading OpenSSL are beyond the scope of this manual.

See the OpenSSL site for more information. OpenSSL downloads can be found on the OpenSSL Source page. Instructions for installing OpenSSL for Jabberd 2 are included in the appendix to this guide. See Installing OpenSSL for Jabberd 2.

2.4.2 Libidn

Libidn provides necessary string manipulation functionality for Jabberd 2. Prior to Jabberd 2 stable 3, libidn was included with the Jabberd 2 distribution; however, a licensing conflict makes it necessary that libidn be installed separately.

N *Note Minimum libidn Version*

Jabberd 2 relies on libidn version 0.3.0 or higher.

See the Libidn site for more information. Libidn downloads can be found on the Libidn Source page. Instructions for installing libidn are included in the appendix to this guide. See Installing Libidn for Jabberd 2.

2.4.3. Data Storage Package

Jabberd 2 has better database integration than was previously supported, and Jabberd 2 can use one of three free databases to provide data storage:

- MySQL
- Berkeley DB (4.1.24 or higher)
- PostgreSQL

MySQL is the recommended and default data store. A file may be used for storing Jabberd 2 data; however, this is not recommended.

N *Note: MySQL and Unicode Support*

MySQL versions 4.1 and above support Unicode (UCS−2 and UTF8) character encoding. If your installation requires support for multiple alphabet encodings, and you wish to use MySQL, choose a version 4.1 or above.

If you have one of these databases installed, you may configure it to work with Jabberd. Otherwise, you should choose one of these databases and install it prior to continuing with Jabberd installation. MySQL is the recommended database; however, Berkeley DB requires the least installation and administration effort. Thus, Berkeley DB may be ideal for an installation with a relatively small number of users.

I *Important: MySQL Requires Development Libraries and Headers*

Note that Jabberd requires more than a minimal MySQL installation. In addition to the basic MySQL installation, Jabberd requires that the development libraries and headers be installed. Either perform a *Max* installation as listed on the MySQL Downloads page, or install *Server*, *Client Programs*, *Libraries and header files*, and *Dynamic client libraries* separately. It may be necessary to uninstall your current MySQL installation in order to install the additional libraries.

Appendices to this document contain instructions for installing either MySQL or Berkeley DB for Jabberd 2. See Installing Berkeley DB for Jabberd 2 or Installing MySQL for Jabberd 2.

2.4.4. Authentication Package

Jabberd 2 can use one of five free third−party authentication data packages to handle user authentication:

- MySQL
- Berkeley DB (version 4.1.24 or higher)
- PostgreSQL

- OpenLDAP (version 2.1.0 or higher)
- PAM

Note that the three supported application data packages can also be used to manage authentication information. Therefore, installing one of MySQL, Berkeley DB or PostgreSQL satisfies both Jabberd data package requirements. A file may be used to store authentication data; however, this is not recommended. The recommended and default package is MySQL.

If you have one of the above five authentication data packages installed, you may configure it for use with Jabberd 2. If not, you should install one or more of the above five packages.

Appendices to this document contain instructions for installing either MySQL or Berkeley DB for Jabberd 2. See Installing Berkeley DB for Jabberd 2 or Installing MySQL for Jabberd 2.

| Back | Up | Next |

3. Install Jabberd 2

This section describes how to build and install Jabberd 2.

3.1. Download Jabberd 2

F *Required File*

Download the file `jabberd-2.0sn.tar.gz` from the Jabber Studio, where "n" is the latest stable version of Jabberd 2.

Download the file referenced above into a directory in `/home` for building the installation files. At the time of writing, Jabberd 2 stable release 3 is the latest version and is used in the examples below.

3.2. Extract Jabberd Installation Files

Change to the directory where you downloaded the file above and then extract the Jabberd 2 files by running the command:

```
tar -zxvf  jabberd-2.0s3.tar.gz
```

3.3. Configure the Jabberd Build

|| TODO: Try to rewrite config instructions to make them clearer. Maybe push some of the special configurations to the FAQ ||

Change to the directory created above:

```
cd jabberd-2.0s3
```

O *Optional: View Configuration Options*

Prior to configuring Jabberd, you can view all configuration options by running the command:

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

This will provide a listing and syntax for setting configuration options. For example, you may choose to install Jabberd into a specific directory by using the `--prefix=PREFIX` option. By providing a `PREFIX` path, all Jabberd files will be installed under this directory. This may be useful if you are testing a new Jabberd installation. Another useful option is `--enable-debug`. This option allows Jabberd to provide detailed debugging information; however, it should be used carefully on production systems.

I *Important: Configuration Options Have Changed with Stable Release 3*
: Jabberd 2 stable 3 introduced changes in the parameters for the `configuration` script. The information below pertains to Jabberd 2 stable 3 and above.

I *Important: Use Explicit Options for OpenSSL and Libidn*
: It is recommended that Jabberd 2 be configured with explicit options for OpenSSL and Libidn. These packages are technically optional. Therefore, `configure` may build Jabberd 2 without them if their libraries and headers are not found when `configure` us run.

P *Parameters: Application Data Package and Authentication Data Package*
: The authentication and storage packages should be specified with the `--enable-` option. Options are `--enable-mysql`, `--enable-pgsql`, `--enable-db`, `--enable-pam` or `--enable-ldap`. Although `--enable-mysql` is the default, it is recommended that this be specified if MySQL is to be used. For example, this command would be used to enable MySQL as the *authentication* and *storage* package without debugging support:

```
./configure --enable-mysql --enable-ssl --enable-idn
```

The following command would be used to configure Jabberd to use Berkeley DB for both storage and authentication and to enable debugging:

```
./configure --enable-db --enable-debug --enable-ssl --enable-idn
```

If your one of the Jabberd prerequisites is installed in a nonstandard location, you will need to specify this location when you run configure. Specify alternate header paths with the `--with-extra-include-path` option and alternate library paths with the `-with-extra-library-path` option. Multiple paths, separated by a colon, may be specified. For example, if OpenSSL and MySQL were installed under `/usr/local`, you might configure with the command:

```
./configure --enable-pgsql --enable-ssl --enable-idn \
--with extra-include-path=/usr/local/include:/usr/local/include/mysql \
--with-extra-library-path=/usr/local/lib:/usr/local/lib/mysql
```

I *Important: Incorrect Parameters Are Ignored*
: Jabberd ignores incorrect configuration parameters. Thus, an incorrectly entered configuration parameter might lead to a successful, however incorrect, Jabberd configuration. Run `./configure --help` if in doubt.

I *Important: Redhat 9 Configuration*
: Building Jabberd 2 on Redhat 9 requires special configuration because Redhat 9 ships with its own version of Kerberos. For more details, see the FAQ about Redhat 9.

If you wish to use the default configuration, simply run the configuration command:

```
./configure
```

This will configure Jabberd to use MySQL and to install to `/usr/local`. If you receive errors, you may wish to check the FAQ where you will find several system-specific work arounds.

3.4. Build Jabberd

Build Jabberd by running the command:

```
make
```

3.5. Install Jabberd

Switch to the super−user:

```
su
```

Run make install:

```
make install
```

3.6. Default File Locations

Your Jabberd 2 installation is complete. Below is a listing of file locations for the default installation:

```
/usr/local/etc/jabberd     Jabberd Configuration Files
/usr/local/bin       Jabberd Binaries (jabberd, c2s, resolver, router, s2s, sm)
```

3.7. Set Ownership of Configuration Files

Jabberd configuration files contain passwords; therefore, you should set ownership and permissions on these files so that they are only readable by your `jabber` user and writable by root only. Using the location of your configuration files and your `jabber` user, set ownership of these files:

```
chown −R root:jabber /usr/local/etc/jabberd/*
```

Then, set permissions on these files so that others can neither read from− or write to them:

```
chmod −R 640 /usr/local/etc/jabberd/*
```

Now, only your `jabber` user and super−user will be able to read and edit your configuration files.

O *Optional: Create Symlink for Configuration Files*

If you used the default file locations when installing Jabberd, you may wish to create a symlink (as superuser) in `/etc` for the configuration files. This will make it easier to find and edit them:

```
ln −s /usr/local/etc/jabberd/ /etc/jabberd
```

Jabberd 2 is now installed. Continue to the next section to being configuring your installation.

Back	Up	Next

4. Basic Configuration

This section provides a quick road map for the most basic configuration and testing of your Jabberd 2 installation. Basic setup for Jabberd 2 consists of these three steps:

1. Set Host Name (sm.xml and c2s.xml)
2. Provision and Configure for Storage and Authentication Package(s)
3. Test Server

Jabberd 2 is configured via its six XML files. For default installations, these configuration files can be found in `/usr/local/etc/jabberd/`, and they are accessible from `/etc/jabberd` if you created the symlink for this directory. Note that this section is easier to complete if you gather the required information in Section 2 beforehand.

4.1. Set Host Name in `sm.xml` and `c2s.xml`

The first step in basic configuration consists of setting the hostname in `sm.xml` and `c2s.xml`.

P *Parameter: Hostname*

Your server hostname (network ID) must be set in both `c2s.xml` and `sm.xml` so that the ID provides a network resolvable reference for your server. In `c2s.xml` this ID is found under the heading labeled `Local network configuration` (approx. line 63), and in `sm.xml` this ID is found under `Session manager configuration` (line 1). Edit `sm.xml` and `c2s.xml` so that this ID references your server.

In `sm.xml`:

```
<!-- Session manager configuration -->
<sm>
  <!-- Our ID on the network. Users will have this as the domain part of
       their JID. If you want your server to be accessible from other
       Jabber servers, this ID must be resolvable by DNS.s
       (default: localhost) -->
  <id>somemachine.somedomain.com</id>
```

In `c2s.xml`:

```
<!-- Local network configuration -->
<local>
  <!-- Who we identify ourselves as. This should correspond to the
       ID (host) that the session manager thinks it is. You can
       specify more than one to support virtual hosts, as long as you
       have additional session manager instances on the network to
       handle those hosts. The realm attribute specifies the auth/reg
       or SASL authentication realm for the host. If the attribute is
       not specified, the realm will be selected by the SASL
       mechanism, or will be the same as the ID itself. Be aware that
       users are assigned to a realm, not a host, so two hosts in the
       same realm will have the same users.
       If no realm is specified, it will be set to be the same as the
       ID. -->
  <id>somemachine.somedomain.com</id>
```

As the `c2s.xml` file notes, this is the hostname that will be appended to your user names to create Jabber ID's, and it must be resolvable via DNS for Jabberd to be accessible via the Internet.

N *Note*

You may use just a domain name (`somedomain.com`) for your Jabberd 2 network ID if your DNS is configured properly to resolve that ID to your server. See Section 5.7. for information about setting up DNS SRV records for Jabberd 2.

4.2. Provision and Configure for Storage and Authentication Package(s)

Getting Jabberd 2 to work with your choice of external *storage* and *authentication* packages involves these steps:

1. Provision external package(s) to work with Jabberd 2
2. Configure `sm.xml` for your choice of *storage* package
3. Configure `c2s.xml` for your choice of *authentication* package

Most Jabberd 2 installations rely on a single package, such as MySQL, to provide both *storage* and *authentication* services. If your installation relies on a single package, you will need to configure this package for Jabberd 2 and then enter similar connection details in both `sm.xml` and `c2s.xml`.

P *Parameters: Data Storage and Authentication Packages*

Follow the guide to complete relevant subsections for your choice(s) of *storage* and *authentication* packages. (Berkeley DB, MySQL or PostgreSQL for storage. Berkeley DB, MySQL, PostgreSQL, PAM or OpenLDAP for authentication.)

Start by jumping to your selection of external *storage* package:

- Provisioning and Configuring for Berkeley DB
- Provisioning and Configuring for MySQL
- Provisioning and Configuring for PostgreSQL

You will then be guided to continue on to provision (if necessary) and configure your *authentication* package. Once your external packages are prepared, and Jabberd is configured to use them, you will guided to Server Testing.

| Back | Up | Next |

4.3. Provision and Configure for Berkeley Berkeley DB

Berkeley DB provides the easiest means to getting your Jabberd 2 server up and running quickly. Jabberd 2 requires a minimum version of 4.1.24 or higher.

4.3.1. Provisioning Berkeley DB

Complete this section if you are using Berkeley DB for storage and/or authorization. Provisioning your system to use Berkeley DB is quite simple. Berkeley DB only needs a directory in which to store data files, and this directory should be owned by the jabber user and group.

P *Parameter: Data Directory*

Create a directory and set permissions (using the user and group created above) for Berkeley DB. (as superuser):

```
mkdir -p /usr/local/var/jabberd/db
chown -R jabber:jabber /usr/local/var/jabberd
```

Berkeley DB is now ready to be used with Jabberd.

4.3.2. Configure for Storage using Berkeley DB (`sm.xml`)

Complete this section if you are using Berkeley DB for storage. Jabberd 2 requires minimal configuration to use Berkeley DB for backend storage. Simply set the driver to use and specify the location for the database.

In `sm.xml` under the section labeled `Storage database configuration`, edit the `driver` to use `db` (Berkeley DB):

```
<!-- Storage database configuration -->
  <storage>
    <!-- By default, we use the MySQL driver for all storage -->
    <driver>db</driver>
```

***P** Parameter: Data Directory*

In `sm.xml` under the section labeled `Berkeley DB driver configuration`, set the path (created in section 4.3.1. above) for your database:

```
<!-- Berkeley DB driver configuration -->
<db>
  <!-- Directory to store database files under -->
  <path>/usr/local/var/jabberd/db</path>

  <!-- Synchronize the database to disk after each write. If you
       disable this, database accesses may be faster, but data may
       be lost if jabberd crashes. -->
  <sync/>
</db>
```

Jabberd 2 is now configured to use Berkeley DB for *storage*.

If you wish to use an *authentication* package other than Berkeley DB, jump to your selection of *authentication* package:

- Provision and Configure for PAM
- Provision and Configure for OpenLDAP

Otherwise, continue on to 4.3.3. directly below to finish your Jabberd 2 configuration.

4.3.3. Configure for Authentication using Berkeley DB (`c2s.xml`)

Complete this section if you are using Berkeley DB for authentication. Jabberd 2 authentication configuration for Berkeley DB is the same as above, except that the information is contained in `c2s.xml`.

In `c2s.xml` under the section labeled `Authentication/registration database configuration`, edit the `module` to use `db` (Berkeley DB):

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>db</module>
```

***P** Parameter: Data Directory*

In `c2s.xml` under the section labeled `Berkeley DB module configuration`, set the path (created in section 4.3.1. above) for your database:

```
<!-- Berkeley DB module configuration -->
<db>
  <!-- Directory to store database files under -->
  <path>/usr/local/var/jabberd/db</path>

  <!-- Synchronize the database to disk after each write. If you
       disable this, database accesses may be faster, but data may
       be lost if jabberd crashes. -->
  <sync/>
</db>
```

Your Jabberd 2 configuration for storage and authentication is now complete. Jump to Test Server to begin testing your server before moving on other configuration tasks, such as configuring SSL, in Section 5.

Back	Up	Next

4.4. Provision and Configure for MySQL

MySQL is the default Jabberd 2 package for storage and authentication.

4.4.1. Provision for MySQL

Complete this section if you are using MySQL for storage and/or authorization. In order to set up MySQL for Jabberd, you must run the setup script included in the Jabberd 2 distribution. After the script is run, you should create a user and then grant that user access to the database.

First, run the MySQL setup script. This script is located in '[Jabberd Source Files]/tools'. Switch to the `tools` directory and start the MySQL console (the MySQL server should already be running). Then, run the `db-setup.mysql` script from the MySQL console:

```
mysql -u root -p
mysql>\. db-setup.mysql
```

Now that a database for Jabberd exists in the MySQL data directory, create a MySQL user that Jabberd can use to connect to the MySQL server.

P *Parameter: MySQL User and Password*

From the MySQL console, run the SQL statement below, replacing `secret` with the password you have chosen for your Jabberd MySQL user:

```
GRANT select,insert,delete,update ON jabberd2.*
to jabberd2@localhost IDENTIFIED by 'secret';
```

Note that the password `secret` is the default password used in the Jabberd configuration files for MySQL.

MySQL is now ready to be used with Jabberd.

N *Note: Default MySQL Socket*

Jabberd 2 stable 3 connects to the MySQL server socket at `/tmp/mysql.sock`. The default socket when installing MySQL from source is `/var/lib/mysql/mysql.sock`. You will need to create a symlink to `/tmp/mysql.sock` if it does not exist:

```
ln −s /var/lib/mysql/mysql.sock /tmp/mysql.sock
```

If you are unsure as to where your MySQL server socket is, consult your MySQL configuration file (usually located in `/etc/my.cnf` or `/etc/mysql/my.cnf`).

4.4.2. Configure for Storage using MySQL (`sm.xml`)

Complete this section if you are using MySQL for storage. Most installations using MySQL for storage will require only the setting of the driver, user and password.

In `sm.xml` under the section labeled `Storage database configuration`, make sure that the `driver` is to use `mysql`. (The driver should be set to mysql by default.):

```
<!−− Storage database configuration −−>
  <storage>
    <!−− By default, we use the MySQL driver for all storage −−>
    <driver>mysql</driver>
```

P *Parameter: MySQL User and Password*

In `sm.xml` under the section labeled `MySQL driver configuration`, replace `secret` with your MySQL password. Change the user if you are not using the default user (`jabberd2`):

```
<!−− MySQL driver configuration −−>
<mysql>
  <!−− Database server host and port −−>
  <host>localhost</host>
  <port>3306</port>

  <!−− Database name −−>
  <dbname>jabberd2</dbname>

  <!−− Database username and password −−>
  <user>jabberd2</user>
  <pass>secret</pass>

  <!−− Transaction support. If this is commented out, transactions
       will be disabled. This might make database accesses faster,
       but data may be lost if jabberd crashes.

       This will need to be disabled if you are using a MySQL
       earlier than v3.23.xx, as transaction support did not appear
       until this version. −−>
  <transactions/>
</mysql>
```

Note that you should change the host setting only if your MySQL server is running on a different host. You should change the port setting only if your MySQL server is running on a non−standard port (port 3306 is the default for MySQL installations). The transaction support section is self−explanatory.

Jabberd 2 is now configured to use MySQL for *storage*.

If you wish to use an *authentication* package other than MySQL, jump to your selection of *authentication* package:

- Provision and Configure for PAM
- Provision and Configure for OpenLDAP

Otherwise, continue on to 4.4.3. directly below to finish your Jabberd 2 configuration.

4.4.3. Configure for Authentication using MySQL (`c2s.xml`)

Complete this section if you are using MySQL for authentication. Jabberd 2 authentication configuration for MySQL is the same as above, except that the information is contained in `c2s.xml`.

***P** Parameter: Authentication Data Package*

In `c2s.xml` under the section labeled `Authentication/registration database configuration`, make sure that the `driver` is to use `mysql`. (The driver should be set to mysql by default.):

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>mysql</module>
```

***P** Parameter: MySQL User and Password*

In `c2s.xml` under the section labeled `MySQL module configuration`, replace `secret` with your MySQL password. Change the user if you are not using the default user (`jabberd2`):

```
<!-- MySQL module configuration -->
<mysql>
  <!-- Database server host and port -->
  <host>localhost</host>
  <port>3306</port>

  <!-- Database name -->
  <dbname>jabberd2</dbname>

  <!-- Database username and password -->
  <user>jabberd2</user>
  <pass>secret</pass>
</mysql>
```

Note that you should change the host setting only if your MySQL server is running on a different host. You should change the port setting only if your MySQL server is running on a non−standard port (port 3306 is the default for MySQL installations).

Your Jabberd 2 configuration for storage and authentication is now complete. Jump to Test Server to begin testing your server before moving on other configuration tasks, such as configuring SSL, in Section 5.

Back	Up	Next

4.5. Provision and Configure for PostgreSQL

PostgreSQL, like MySQL provides a very manageable backend for storage and authentication. Unlike MySQL, PostgreSQL provides better unicode support.

4.5.1. Provision PostgreSQL

Complete this section if you are using PostgreSQL for storage and/or authorization. In order to set up PostgreSQL for Jabberd, you must create a database, create a PostgreSQL user and then run the PostgreSQL

script included in the Jabberd 2 distribution.

Create the database for Jabberd. (The PostgreSQL server should already be running):

```
createdb -U postgres jabberd2
```

The command above will create a database from which you will be able to run the script for setting up the Jabberd PostgreSQL database.

***N** Note: Unicode Support*

If you want to enable Unicode support for your PostgreSQL database, change the command above to the following:

```
createdb -U postgres -E UNICODE jabberd2
```

Now that your database is created, create a PostgreSQL user for the database.

***P** Parameter: PostgreSQL User and Password*

To create your Jabberd database user, enter the command below:

```
createuser -P -U postgres jabberd2
```

This command will initiate an interactive user creation script. When prompted, enter the password that Jabberd will use to connect to your PostgreSQL database:

```
Enter password for user "jabberd2":
Enter it again:
Shall the new user be allowed to create databases? (y/n) n
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

The `CREATE USER` statement indicates that the command was successful.

After your jabberd user is created, you are ready to run the the PostgreSQL setup script. This script is located in '[Jabberd Source Files]/tools'. Switch to the `tools` directory and start the PostgreSQL console as the jabberd2 user:

```
psql -U jabberd2 jabberd2
```

Then, run the `db-setup.pgsql` script from the PostgreSQL console:

```
jabberd2=>\i db-setup.pgsql
```

PostgreSQL is now ready to be used with Jabberd 2.

4.5.2. Configure for Storage using PostgreSQL (`sm.xml`)

Complete this section if you are using PostgreSQL for storage. Most installations using PostgreSQL for storage will require only the setting of the driver, user and password.

In `sm.xml` under the section labeled `Storage database configuration`, edit the `driver` to use pgsql (PostgreSQL):

```
<!-- Storage database configuration -->
  <storage>
    <!-- By default, we use the MySQL driver for all storage -->
    <driver>pgsql</driver>
```

***P** Parameter: PostgreSQL User and Password*

In `sm.xml` under the section labeled `PostgreSQL driver configuration`, replace `secret` with your PostgreSQL password. Change the user if you are not using the default user (`jabberd2`):

```
<!-- PostgreSQL driver configuration -->
<pgsql>
  <!-- Database server host and port -->
  <host>localhost</host>
  <port>5432</port>

  <!-- Database name -->
  <dbname>jabberd2</dbname>

  <!-- Database username and password -->
  <user>jabberd2</user>
  <pass>secret</pass>

  <!-- Transaction support. If this is commented out, transactions
       will be disabled. This might make database accesses faster,
       but data may be lost if jabberd crashes. -->
  <transactions/>
</pgsql>
```

Note that you should change the host setting only if your PostgreSQL server is running on a different host. You should change the port setting only if your PostgreSQL server is running on a non−standard port (port 5432 is the default for PostgreSQL installations). The transaction support section is self−explanatory.

Jabberd 2 is now configured to use PostgreSQL for *storage*.

If you wish to use an *authentication* package other than PostgreSQL, jump to your selection of *authentication* package:

- Provision and Configure for PAM
- Provision and Configure for OpenLDAP

Otherwise, continue on to 4.5.3. directly below to finish your Jabberd 2 configuration.

4.5.3. Configure for Authentication using PostgreSQL (`c2s.xml`)

Complete this section if you are using PostgreSQL for authentication. Jabberd 2 authentication configuration for PostgreSQL is the same as above, except that the information is contained in `c2s.xml`.

In `c2s.xml` under the section labeled `Authentication/registration database configuration`, edit the `module` to use pgsql (PostgreSQL):

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>pgsql</module>
```

P *Parameter: PostgreSQL User and Password*

In `c2s.xml` under the section labeled `PostgreSQL module configuration`, replace `secret` with your PostgreSQL password. Change the user if you are not using the default user (`jabberd2`):

```
<!-- PostgreSQL module configuration -->
<pgsql>
  <!-- Database server host and port -->
  <host>localhost</host>
  <port>5432</port>

  <!-- Database name -->
  <dbname>jabberd2</dbname>

  <!-- Database username and password -->
  <user>jabberd2</user>
  <pass>secret</pass>
</pgsql>
```

Note that you should change the host setting only if your PostgreSQL server is running on a different host. You should change the port setting only if your PostgreSQL server is running on a non-standard port (port 5432 is the default for PostgreSQL installations).

Your Jabberd 2 configuration for storage and authentication is now complete. Jump to Test Server to begin testing your server before moving on other configuration tasks, such as configuring SSL, in Section 5.

Back	Up	Next

4.6. Provision and Configure for PAM

PAM (Pluggable Authentication Modules for Linux) provides built in authentication support for Jabberd 2.

4.6.1. Provision for PAM

Complete this section if you are using PAM for authorization. Authentication via PAM requires a valid PAM configuration file named `jabberd`. For many systems, this configuration file should be located under `/etc/pam.d`. Creation of Jabberd PAM configuration file is beyond the scope of this guide; however, a shortcut may be used to create this configuration file. Copy the `system-auth` configuration file to `jabberd` (as root):

```
cp /etc/pam.d/system-auth /etc/pam.d/jabberd
```

This will create a PAM configuration file that can be used by Jabberd2.

I *Important: PAM Authentication Requires PAM Database Access*

Jabberd authentication via PAM requires that Jabberd2 has access to the PAM database. For many systems, this database is the `/etc/shadow` file. Thus, Jabberd2 must be run as root, or the `jabberd` user must be granted read permissions for this file. Running the Jabberd2 server as root is not recommended.

N *Note: Winbind Can Be Used to Integrate NT Authentication with PAM*

The Winbind package from the Samba project can be used to support Jabber authentication via Windows NT. With Winbind successfully installed and configured, configure Jabberd to use PAM for

authentication. Then create the `jabberd` PAM configuration file as below:

```
auth       required     pam_winbind.so
password   required     pam_winbind.so
account    required     pam_winbind.so
session    required     pam_winbind.so
```

PAM is now ready to be used with Jabberd 2. Continue on to begin configuring Jabberd 2 to authenticate against PAM.

4.6.2. Configure for Authentication using PAM (`c2s.xml`)

Complete this section if you are using PAM for authentication. Jabberd requires little configuration to use PAM.

In `c2s.xml` under the section labeled `Authentication/registration database configuration`, edit the `module` to use pam:

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>pam</module>
```

Users cannot create their own accounts when using PAM for authentication. Therefore, public account registration should be disabled, while `auto-create` should be enabled so that the session manager can create accounts the first time users log on.

In `c2s.xml`, look for the `Registration configuration` subsection under the `Authentication/registration database configuration` section. Commenting the `enable` tag as below will disable public registration:

```
<!-- Registration configuration -->
<register>
  <!-- Account registration is enabled by default (provided the
       auth/reg module in use supports it). Comment this out to
       disable. -->
  <!-- <enable/> -->
```

In `sm.xml` under the section labeled `User options` (near the bottom of the file), uncomment the `auto-create` tag as below so that the session manager will create a new Jabberd2 account the first time a user logs on:

```
<!-- User options -->
<user>
  <!-- By default, users must explicitly created before they can start
       a session. The creation process is usually triggered by a c2s
       component in response to a client registering a new user.

       Enableing this option will make it so that a user create will be
       triggered the first time a non-existant user attempts to start
       a session. This is useful if you already have users in an
       external authentication database (eg LDAP) and you don't want
       them to have to register. -->
  <auto-create/>
```

Your Jabberd 2 configuration for storage and authentication is now complete (provided that you have provisioned and configured for a storage package). Jump to Test Server to begin testing your server before moving on other configuration tasks, such as configuring SSL, in Section 5.

|| TODO: note that PAM should use SSL/TLS ||

| Back | Up | Next |

4.7. Provision and Configure for OpenLDAP

OpenLDAP provides authentication support that is distributable across platforms and geography. Jabberd 2 requires a minimum version of OpenLDAP 2.1.0.

4.7.1. Provision for OpenLDAP

Complete this section if you are using OpenLDAP authorization. Your OpenLDAP installation should not require special configuration for Jabberd 2; however, at the time of writing (Jabberd 2 stable release 3), there is an important issue regarding Jabberd connection with OpenLDAP v3 servers. Jabberd 2 currently uses v2 syntax. By default, OpenLDAP v3 servers require v3 syntax.

There is a workaround for this issue. Add the following statement to your slapd.conf file, and restart your slapd daemon:

```
allow bind_v2
```

This statement will allow your Jabberd 2 server to connect to your OpenLDAP v3 server.

4.7.2. Configure for Authentication using OpenLDAP (`c2s.xml`)

Complete this section if you are using OpenLDAP for authentication. OpenLDAP configuration is more detailed because configuration requires host and connection settings in addition to query settings.

In `c2s.xml` under the section labeled `Authentication/registration database configuration`, edit the `driver` to use ldap (OpenLDAP):

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>ldap<module>
```

P *Parameter: LDAP Connection Settings*

The first part of the `LDAP module configuration` in `c2s.xml` deals with the settings required to connect to your OpenLDAP server. The `host` must either be a hostname resolvable by the server or the IP address of the OpenLDAP server. Port 389 is the default port for OpenLDAP servers, so in most cases, `port` should be left as is. The `v3` tag specifies whether your OpenLDAP server is v3. Uncomment this tag if it is. Leave the `v3` tag commented for OpenLDAP v2 servers. Lastly, uncomment either the `starttls` or `ssl` tag if your server supports encryption (see notes below):

```
<!-- LDAP module configuration -->
<ldap>
```

```
<!-- LDAP server host and port (default: 389) -->
<host>ldap.example.com</host>
<port>389</port>

<!-- Use LDAP v3 if possible. If disabled, v2 will be used.
     Encryption options are only available if v3 is enabled. -->
<!--
<v3/>
-->

<!-- Encryption. If enabled, this will create an encrypted channel
     to the LDAP server using the LDAP STARTTLS mechanism. -->
<!--
<starttls/>
-->

<!-- Encryption. If enabled, this will create an encrypted channel
     to the server using the old-style "ldaps://" mechanism. It is
     recommended that you use <starttls/> instead of this. -->
<!--
<ssl/>
-->
```

I *Important: Passwords are Transmitted in Clear Text*

The current release of Jabberd 2 (stable 3) requires that plain text passwords be used when authenticating via OpenLDAP. Therefore, it is extremely important that you enable encryption. As noted above, encryption options are available only for connection with OpenLDAP v3 servers.

|| TODO: note that PAM should use SSL/TLS ||

N *Note: Encryption Options*

The author has had success using Jabberd 2 and STARTTLS with a recent version of the OpenLDAP server. STARTTLS runs over the standard port (389), so all that needs to be done to enable STARTTLS is to uncomment the `v3` and `starttls` tags. On the other hand, the author has not had success using SSL encryption between Jabberd 2 and OpenLDAP. To set up SSL, you should specify the OpenLDAP SSL hostname (if different from the non-SSL hostname). This is often something like `ldaps.example.com`. More importantly, you should specify the SSL port. The standard SSL port (LDAPS) for OpenLDAP is 636. Lastly, you should uncomment the `ssl` tag.

P *Parameter: OpenLDAP User and Password*

The next part of the OpenLDAP configuration handles the user and password for queries on the OpenLDAP server. A user and password are required ***only*** if your OpenLDAP server does ***not*** permit anonymous binding (authentication) for the required searches. Uncomment this section if your OpenLDAP server requires authentication for connection. Note that the `binddn` is the full RDN (relative distinguished name) for the user. This may be something like `cn=admin,ou=people,dc=example,dc=com`:

```
<!-- DN to bind as for searches. If unspecified, the searches
     will be done anonymously. -->
<!--
<binddn>cn=Directory Manager</binddn>
<bindpw>secret</bindpw>
-->
```

P *Parameter: LDAP Query Settings*

Lastly, the user ID and base DN must be set for OpenLDAP queries. User ID is specified by the `uidattr` tags. This ID should be the attribute by which your users are uniquely identified under the specified base DN (distinguished name). In other words, when querying against the specified base DN, the specified `uidattr` should uniquely identify each user. The `basedn` attribute specifies the

base against which queries are run. This can be the topmost DN of the OpenLDAP server, such as `dc=example,dc=com` or it can be an RDN below which the user entries are found, such as `ou=people,ou=sales,dc=example,dc=com`. Using a lower level RDN is likely to speed OpenLDAP queries. Lastly, if your configuration requires multiple realms, you can specify a base DN for each by using the `realm` attribute of the `basedn` tag. Note that if you are not using multiple realms, you need only to specify a single `basedn` without the `realm` attribute:

```
      <!-- LDAP attribute that holds the user ID (default: uid) -->
      <uidattr>uid</uidattr>

      <!-- base DN of the tree. You should specify a DN for each
           authentication realm declared in the <local/> section above,
           by using the realm attribute. -->
      <basedn realm='company'>o=Company.com</basedn>
      <basedn>o=Example Corp.</basedn>
    </ldap>
```

Below is an obfuscated view of the author's working OpenLDAP configuration:

```
<!-- LDAP module configuration -->
<ldap>
  <!-- LDAP server host and port (default: 389) -->
  <host>ldap.mydomain.org</host>
  <port>389</port>

  <!-- Use LDAP v3 if possible. If disabled, v2 will be used.
       Encryption options are only available if v3 is enabled. -->

  <v3/>

  <!-- Encryption. If enabled, this will create an encrypted channel
       to the LDAP server using the LDAP STARTTLS mechanism. -->

  <starttls/>

  <!-- Encryption. If enabled, this will create an encrypted channel
       to the server using the old-style "ldaps://" mechanism. It is
       recommended that you use <starttls/> instead of this. -->
  <!--
  <ssl/>
  -->

  <!-- DN to bind as for searches. If unspecified, the searches
       will be done anonymously. -->

  <!--
  <binddn>cn=admin,dc=mydomain,dc=org</binddn>
  <bindpw>snip</bindpw>
  -->

  <!-- LDAP attribute that holds the user ID (default: uid) -->
  <uidattr>uid</uidattr>

  <!-- base DN of the tree. You should specify a DN for each
       authentication realm declared in the <local/> section above,
       by using the realm attribute. -->

  <basedn>ou=people,ou=design,dc=mydomain,dc=org</basedn>
</ldap>
```

Users cannot create their own accounts when using OpenLDAP for authentication. Therefore, public account registration should be disabled, while `auto-create` should be enabled so that the session manager can create accounts the first time users log on.

In `c2s.xml`, look for the `Registration configuration` subsection under the `Authentication/registration database configuration` section. Commenting the `enable` tag as below will disable public registration:

```
<!-- Registration configuration -->
<register>
  <!-- Account registration is enabled by default (provided the
       auth/reg module in use supports it). Comment this out to
       disable. -->
  <!-- <enable/> -->
```

In `sm.xml` under the section labeled `User options` (near the bottom of the file), uncomment the `auto-create` tag as below so that the session manager will create a new Jabberd2 account the first time a user logs on:

```
<!-- User options -->
<user>
  <!-- By default, users must explicitly created before they can start
       a session. The creation process is usually triggered by a c2s
       component in response to a client registering a new user.

       Enableing this option will make it so that a user create will be
       triggered the first time a non-existant user attempts to start
       a session. This is useful if you already have users in an
       external authentication database (eg LDAP) and you don't want
       them to have to register. -->
  <auto-create/>
```

Your Jabberd 2 configuration for storage and authentication is now complete (provided that you have provisioned and configured for a storage package). Jump to the next section, Test Server, to begin testing your server before moving on other configuration tasks, such as configuring SSL, in Section 5.

Back	Up	Next

4.8. Test Server

After setting the hostname, provisioning external package(s), and configuring Jabberd to use your external package(s), your server is ready for testing.

C *Checkpoint: Start Your Server*

You should be able to start and test your Jabberd 2 server by using the Jabberd 2 startup script (as your jabber user):

```
su
su jabber
cd /usr/local/bin
./jabberd
```

N *Note: Troubleshooting*

If Jabberd does not start, make sure that any previous instances have stopped. These instances include all the Jabberd runtime components (`jabberd`, `router`, `resolver`, `sm`, `s2s` and `c2s`). Note that your `jabber` user (if created according to section 2.) may not have default PATH's; therefore, you should `cd` to the `/usr/local/bin` directory and run `jabberd` as above. Check that your chosen data package servers are running (except Berkeley DB, which does not require starting). Check your `syslog` for error messages. If your server fails to start, you can start Jabberd 2 with the debug option (note that this requires building Jabberd 2 with the debug option see Section 3.3):

```
/usr/local/bin/jabberd -D
```

I *Important: Jabberd 2 Should Not Be Run as Superuser*

As with all daemons, Jabberd 2 should not be run as superuser. Running Jabberd 2 as super−user not only risks damage to the system, but also running as super−user may create file permission problems.

N *Note: Public Registration*

Public registration for new users is enabled in Jabberd2 by default. Thus, when testing your server, you can create a new user by logging on as a new user.

C *Checkpoint: Connect from a Machine on the Same Network*

Once you have verified that your server is starting correctly, try connecting from a machine on the same network. In a Jabber client, enter a JID (Jabber ID) that uses the ID you set in `sm.xml` in Section 4.1, and try to connect to your server.

C *Checkpoint: Connect from Client on Remote Network*

If you are able to connect to your server on the same network, use a Jabber client to connect to it from a remote network. This will test that `id` is set properly and that the machine name is resolvable via DNS.

Your Jabberd 2 server is now ready to use. Continue to Common Configuration Tasks detailed configuration options, such as enabling SSL connections.

Back	Up	Next

5. Common Configuration Tasks

Before delving into detailed Jabberd configuration, this section attempts to provide a guide for the most common Jabberd configuration tasks:

- Configuring Firewall for Internet Access
- Configuring Jabberd 2 for SSL
- Changing Router Password
- Creating an Administrative User
- Disabling Public Registration
- Enabling User Password Change
- Setting DNS SRV Records
- Using Jabberd 1.4 to Connect to Legacy Services
- Using JCR to Connect to Legacy Services
- Setting up a JUD Using Users−Agent
- Integrating Users−Agent with vCard Data (private servers only)

Note that there are two options for connecting Jabberd 1.4 legacy services to your Jabberd 2 installation. Jabberd 2 can connect to services, such as conferencing and gateways, running within a Jabberd 1.4 process. Additionally, a component wrapper called JCR has been released, and this wrapper allows a Jabber 1.4

component (written in C) to be compiled and run as Jabberd 2 service. At the time of writing, JCR has been tested with MU Conferencing only.

5.1. Configuring Firewall for Internet Access

Although firewall configuration is beyond the scope of this guide, administrators should be aware of the TCP ports that need to be enabled for Internet access:

- port 5222 *non−SSL client communication*
- port 5223 *SSL client communication*
- port 5269 *server to server communication*

TCP ports should be enabled as above and according to your configuration if your Jabberd 2 installation is to access the Internet.

5.2. Configuring Jabberd 2 for SSL Connections

Jabberd 2 is designed to provide for STARTTLS and SSL connections not only between Jabber clients and the server, but also between the Jabberd server components (`sm`, `resolver`, `s2s` and `c2s`) and the Jabberd `router`. A single SSL certificate may be used for these two functions (Jabber client to Jabberd and Jabberd component to router), or two separate keys may be used. See the appendix, Generating a Self−Signed SSL Certificate for instructions about how to create your own self−signed certificate for use by Jabberd 2.

N *Note: Self−Signed Certificates Not Trusted*

Note that self−signed certificates are not automatically trusted by Jabber clients because there is no chain of authority against which to verify authenticity. Nevertheless, creating a self−signed certificate not only will allow your Jabber users to communicate over a secure channel (possibly with warnings displayed by the client), but also such a certificate will provide for secure communication among the five Jabberd components (`router`, `sm`, `resolver`, `s2s` and `c2s`).

N *Note: Adding Certificates to the PSI Jabber Client*

I have written a shell script that retrieves and adds a self−signed certificate to the PSI client certificate store. It is available here in the tools folder.

5.2.1. Assigning a Certificate for Use by Jabber Clients

The SSL key for Jabber clients is located in `c2s.xml`. Note that `c2s.xml` contains the location of the SSL key used by Jabber clients in addition to the location of the SSL key used for `c2s` to `router` communications.

P *Parameter: SSL Key Location*

Uncomment the `pemfile` (your SSL key) location under the section labeled `Local network configuration`, and edit it for the location of your SSL key. Note that if your PEM file is in the default location of `/usr/local/etc/jabberd/server.pem`, you need only uncomment this section as below:

```
<!−− File containing a SSL certificate and private key for client
     connections. If this is commented out, clients will not be
     offered the STARTTLS stream extension −−>

<pemfile>/usr/local/etc/jabberd/server.pem</pemfile>
```

The change above will enable STARTTLS on port 5222. Older Jabber clients use port 5223 for SSL enabled communications. If you wish to support SSL on port 5223, uncomment the `ssl-port` tags:

```
<ssl-port>5223</ssl-port>
```

Your server is now ready for STARTTLS/SSL connections. You need only restart the C2S component for the SSL change to take effect.

***N** Note: Disabling Non-STARTTLS Communication*

To require STARTTLS communications, uncomment the `require-starttls` tag as below:

```
<!-- Require STARTTLS. If this is enabled, clients must do STARTTLS
     before they can authenticate. Until the stream is encrypted,
     all packets will be dropped. -->

<require-starttls/>
```

5.2.2. Assigning a Certificate for Use by Jabberd Components

Each of the five Jabberd components has its own configuration for encrypted component-to-router communications. Thus, these five configuration files must be edited to provide secure communication among Jabberd components:

```
router.xml
sm.xml
resolver.xml
s2s.xml
c2s.xml
```

***P** Parameter: SSL Key Location*

In each of the files above, uncomment the `pemfile` tag for router communications. In the `router.xml` file, the `pemfile` is specified under the section labeled `Local network configuration`. In each of the the remaining four configuration files above, the `pemfile` location is specified under the section labeled `Router connection configuration`. Uncomment this section and edit it to point to the location of your SSL key. For example, you would edit `c2s.xml` as below if you are using the default location for your SSL key:

```
<!-- Router connection configuration -->
<router>
  <!-- IP/port the router is waiting for connections on -->
  <ip>127.0.0.1</ip>      <!-- default: 127.0.0.1 -->
  <port>5347</port>       <!-- default: 5347 -->

  <!-- Username/password to authenticate as -->
  <user>jabberd</user>    <!-- default: jabberd -->
  <pass>secret</pass>     <!-- default: secret -->

  <!-- File containing a SSL certificate and private key to use when
       setting up an encrypted channel with the router. If this is
       commented out, or the file can't be read, no attempt will be
       made to establish an encrypted channel with the router. -->

  <pemfile>/usr/local/etc/jabberd/server.pem</pemfile>
```

Restart your Jabberd server for the change to take effect.

|| TODO: change infor about disabling non−ssl comms. Add info about requiring TLS ||

5.3. Changing Router Password

The Jabberd configuration files contain passwords used to connect to the *router* component because communications between components and the router occur by XML over TCP/IP. These passwords help to ensure that only your installed components can communicate with the *router*. The configuration file, `router−users.xml`, contains ID's and password(s) for components that are allowed to connect to the router. By default, the ID is `jabberd` and the password is `secret`.

Additionally, each of the components (except the *router*) has a user ID and password specified in its configuration file. This is the ID and password combination that the respective component uses to connect with the router. Thus, for a component to be able to connect to the router, the component must have a user and password pair specified in its configuration file, and that pair must match an ID and password pair in `router−users.xml`.

To improve security for your Jabberd installation, you should change the password. This involves changing the password in `router−users.xml` and then in `sm.xml`, `resolver.xml`, `s2s.xml` and `c2s.xml`.

P *Parameter: Router User and Password*

In order to change the password used for authentication by the router, first change the password in `router−users.xml` as below (replacing `newpass` with your new password):

```
<users>
  <user>
    <name>jabberd</name>
    <secret>newpass</secret>
  </user>
</users>
```

Then, change the password in each of `sm.xml`, `resolver.xml`, `s2s.xml` and `c2s.xml`. In each of these files there are tags for the *router user* in the `router` section. Change the password as below (replacing `newpass` with your new password):

```
<!−− Router connection configuration −−>
<router>
  <!−− IP/port the router is waiting for connections on −−>
  <ip>127.0.0.1</ip>      <!−− default: 127.0.0.1 −−>
  <port>5347</port>       <!−− default: 5347 −−>

  <!−− Username/password to authenticate as −−>
  <user>jabberd</user>    <!−− default: jabberd −−>
  <pass>newpass</pass>     <!−− default: secret −−>
```

Restart your Jabberd server for the change to take effect.

N *Note: Multiple Passwords Allowed*

Note that you can assign a user and ID for each of the components if you wish.

N *Note:* `router.xml` *Must Be Edited If User Is Changed*

Note that the above edits describe how to change the router password only. If the user name in `router−users.xml` is changed, then the user listed in the `acl` section of `router.xml` must also be changed:

```
<!−− Access control information −−>
```

```
<aci>
  <!-- The usernames listed here will get access to all restricted
       functions, regardless of restrictions further down -->
  <acl type='all'>
    <user>jabberd</user>
  </acl>
```

5.4. Creating an Administrative User

Settings for administrative users are contained in the `aci` section of `sm.xml`. By default, the administrative user is `admin@localhost`. In order to enable an administrative user that can be accessed remotely, change the `jid` to a user of your own choosing as below:

```
<acl type='all'>
  <jid>admin@lsomedomain.com</jid>
</acl>
```

You will also need to create this user manually or from a Jabber client. When logged in, the administrative user will receive notices for user creation, and the administrative user will also be able to discover all online users, receive help requests, send MOTD's (messages of the day), etc.

Note that the user above is granted access to *all* administrative functions. You can assign specific administrative functions to users by specifying the `acl type`. See the examples in the `sm.xml` file.

Restart your Jabberd server for the change to take effect.

5.5. Disabling Public Registration

By default, Jabberd allows public registration for all users, which is to say that any user who can connect to your server can create their own Jabberd user on your server. In order to prevent public registration, edit the `c2s.xml` configuration file.

Under the `Authentication/registration database configuration` section, look for the `Registration configuration` subsection. Commenting the `enable` tag as below will disable public registration:

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>mysql</module>

  <!-- Registration configuration -->
  <register>
    <!-- Account registration is enabled by default (provided the
         auth/reg module in use supports it). Comment this out to
         disable. -->
    <!--
    <enable/>
    -->
```

Configurations using either PAM or OpenLDAP should disable public registration because Jabberd 2 does not support public registration via PAM or OpenLDAP.

5.6. Enabling User Password Change

User password changing is disabled by default in Jabberd 2. To allow your users to change their own passwords, uncomment the `password` tag in the `authreg` section in `c2s.xml` as below:

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>mysql</module>

  <!-- Registration configuration -->
  <register>
    <!-- Account registration is enabled by default (provided the
         auth/reg module in use supports it). Comment this out to
         disable. -->
    <enable/>

    <!-- Human-readable instructions to be returned to client when
         registration is requested. -->
    <instructions>Enter a username and password to register with this server.</instructions

    <!-- Password change only. When registration is disabled, it may
         still be useful to allow clients to change their password. If
         you want this, uncomment this when you disable registration. -->
    <password/>
```

Note that this will have no effect when using PAM or OpenLDAP for authentication because these packages do not permit password changing via Jabberd. Restart your Jabberd server for the change to take effect.

5.7. Setting DNS SRV Records

Jabberd 2, in addition to other Jabber clients and servers, is able to use DNS SRV records for hostname resolution. DNS SRV records allow for delegation of services by port to other hosts. Thus, if you want your Jabber server to run on a host that is not the primary domain host, you would most likely want to set DNS SRV records to delegate Jabber client and server services to another host or hosts.

***N** Note: SRV Records Required Only for Non-Primary Host*

Note that DNS SRV records are required only if your Jabberd server is running on a host other than the primary domain host and if you do not wish to include the host (machine) name in your Jabber ID. For example, if a DNS query for `somedomain.com` resolves `host1.somedomain.com`, and your Jabberd server is running on `host1`, SRV records would not be required.

5.7.1. SRV Records for Jabberd

There are 3 SRV records that can be created for a Jabberd installation:

```
_jabber._tcp.<domain>   ->    <host>.<domain>:5269
_xmpp-server._tcp.<domain>    ->    <host>.<domain>:5269
_xmpp-client._tcp.<domain>    ->    <host>.<domain>:5222
```

The first and second of these specify the host and the port for server-to-server (s2s) communications. There are two listings for this because the new XMPP protocol, regarding SRV records, is replacing the older Jabber standards. The third listing above specifies host and port for unencrypted client communications (c2s).

5.7.2. Creating SRV Records in Bind

The following are examples for creating a set of SRV records for the BIND server:

```
_jabber._tcp.some_domain.com. 86400 IN SRV 5 0 5269 host.some_domain.com.
_xmpp-server._tcp.some_domain.com. 86400 IN SRV 5 0 5269 host.some_domain.com.
_xmpp-client._tcp.some_domain.com. 86400 IN SRV 5 0 5222 host.some_domain.com.
```

Replace `some_domain.com` with your domain name and `host` with the name of the host, and do not omit the "." after the domain name.

5.7.3. Creating SRV Records in TinyDNS

TinyDNS does not have a format for SRV records; however, you can use Rob Mayoff's TinyDNS Record Maker to create TinyDNS SRV records. These TinyDNS SRV records were created for the host of `host.some_domain.com` using a priority of `10` and a weight of '0':

```
:_jabber._tcp.some_domain.com:33:\000\012\000\000\024\225\004host\013some_domain\003c
:_xmpp-client._tcp.some_domain.com:33:\000\012\000\000\024\146\004host\013some_domain
:_xmpp-server._tcp.some_domain.com:33:\000\012\000\000\024\225\004host\013some_domain
```

Use TinyDNS Record Maker to create a set of records to be added to the TinyDNS data file.

5.7.4. Testing SRV Records

Once the your DNS server is properly updated, you should test the listings using Dig. For example, to test the entry of `_jabber._tcp.some_domain.com`, using the DNS server `my.dns_server.com`, you would enter the command below:

```
dig @my.dns_server.com _jabber._tcp.some_domain.com any +short
```

This should provide you with the data from your DNS SRV record:

```
10 0 5269 host.some_domain.com.
```

5.8. Using Jabberd 1.4 to Connect to Legacy Services

This section describes how to use an existing Jabberd 1.4 installation to connect legacy Jabberd 1.4 services, such as gateways and transports, to Jabberd 2. See the next section for using the JCR component to connect to legacy services. See Appendix 11 for a primer on how Jabberd 2 can use a Jabberd 1.4x process to run a transport.

A familiarity with Jabberd 1.4. is assumed, as is a working Jabberd 1.4 installation. Jabberd 1.4 setup and configuration is beyond the scope of this document; however, many excellent resources exist, including the Jadmin Archive and the Jabberd 1.4x Administration Guide.

Connecting Jabberd to a legacy service is similar to the means by which Jabberd 1.4 links to services; however there are several important differences when linking from Jabberd 2 to a Jabberd 1.4 service:

- The linked configuration file must have an XDB section
- The linked configuration file must have a log section
- The linked configuration file must specify the router IP address and port
- The linked configuration file and the router configuration file must share a password

- The router.xml file must contain an alias for the service

Additionally, a service namespace should exist in `sm.xml` if the component does not support discovery (`disco`). For example, MU Conference supports discovery, so no entry is required in `sm.xml` in order to make this service browsable. On the other hand, JUD (Jabber User Directory) does not support discovery; therefore, an entry would be required in `sm.xml` to make a JUD browsable by users. See `sm.xml` for examples.

The sub−sections below describe how to configure a David Sutton's MU Conference for Jabberd 1.4, and the example concludes with a working MU Conference configuration file. See MU Conference for downloads and more information about this component. MU Conference provides a multi−user chat room service for Jabberd.

5.8.1. XDB Section

When running a gateway or transport with Jabberd 1.4, the main Jabberd process handles XDB and logging functions. This is not the case when running such a component with Jabberd 2. Instead, the component should run in its own process, and the component should handle its own XDB and logging functions. That is to say that legacy components should be run in a process that is self−contained. Thus, if you have a component that is linked to a Jabberd 1.4. server, you should add an *XDB* section to the service configuration file:

```
<xdb id="xdb">
  <host>conference.somedomain.com</host>

  <load>
    <xdb_file>/usr/local/jabber/xdb_file/xdb_file.so</xdb_file>
  </load>

  <xdb_file xmlns="jabber:config:xdb_file">
    <spool>/usr/local/var/spool/jabber</spool>
  </xdb_file>
</xdb>
```

The *XDB* section above specifies the hostname, the XDB module to load, and the location to write the spool file(s).

5.8.2. Log Section

As noted above, the component configuration file must also contain a logging section. You might add a section like this to the configuration file for your linked component:

```
<log id="muclog">
  <file>/usr/local/var/jabberd/log/muc.log</file>
  <host/>
  <logtype/>
  <format>%d: [%t] (%h): %s</format>
</log>
```

In the case of an MU Conference gateway, I have specified the log above as `muc.log`. Each service should have a separate log file.

5.8.3. Router IP Address and Port

The linked component should specify the IP address and port that the *router* is listening on. The default port for the *router* is `5347`, so the beginning of the `id` section of your linked file would look something like this:

```
<service id="muclinker">
  <uplink/>
  <connect>
    <ip>192.168.0.2</ip> <!-- IP Address of Router here -->
    <port>5347</port>
```

5.8.4. Shared Password

The Jabberd 2 server uses a password for component connections. This password is similar to the *shared secret* that is used in Jabberd 1.4. Continuing with the example above, you should create a password for your legacy component to use. (In Jabberd 2, all legacy components share the same password with the *router*) This password, or `secret`, must be specified in your the configuration file for your linked component:

```
<service id="muclinker">
  <uplink/>
  <connect>
    <ip>192.168.0.2</ip> <!-- IP Address of Router here -->
    <port>5347</port>
    <secret>ComponentPass</secret>
  </connect>
</service>
```

This `secret` must also be specified in your `router.xml` file. The `secret` is set in the section labeled 'local network configuration':

```
<!-- Local network configuration -->
<local>
  <!-- IP address to bind to (default: 0.0.0.0) -->
  <ip>0.0.0.0</ip>
  <!-- Port to bind to (default: 5347) -->
  <port>5347</port>
  <!-- File containing the user table. This is where the router gets
       its component and secret information from for component
       authentication.-->
  <users>/usr/local/etc/jabberd/router-users.xml</users>
  <!-- Shared secret used to identify legacy components (that is,
       "jabber:component:accept" components that authenticate using
       the "handshake" method). If this is commented out, support for
       legacy components will be disabled. -->
  <secret>ComponentPass</secret>
```

Note that the password is specified with the `secret` tag in `router.xml` (as above), and not in the `router-users.xml` file.

5.8.5. Router Name Alias

The `router.xml` file must also contain an alias for the linked component, and the `router.xml` file provides an example for uplinking an MSN transport. Add an alias section for the component to which you are linking. Continuing with the example above, this alias would be set as below in 'router.xml':

```
<!-- Name aliases.

     Packets destined for the domain specified in the "name" attribute
     will be routed to the component that has currently bound the name
     in the "target" attribute (assuming it is online).

     This is usually only required for some kinds of legacy
     components (particularly jabberd 1.4 "uplink" components) -->
```

```
<aliases>
  <!-- Example for a msn transport running from a jabberd 1.4 uplink -->
  <!--
  <alias name='msn.domain.com' target='msn-linker'/>
  -->
  <alias name='conference.somedomain.com' target='muclinker'/>
</aliases>
```

Note that the alias references the DNS-resolvable name for the component, and the `target` references the `service id` as specified in the linked component XML file (as shown in Section 5.5.4).

5.8.6. Example: *muc.xml*

A working `muc.xml` file for MU Conferencing appears below:

```
<jabber>

  <service id="muclinker">
    <uplink/>
    <connect>
      <ip>192.168.0.2</ip> <!-- IP Address of Router here -->
      <port>5347</port>
      <secret>ComponentPass</secret>
    </connect>
  </service>

  <service id="conference.somedomain.com">

    <load>
       <conference>/usr/local/jabber/mu-conference/src/mu-conference.so</conference>
    </load>

    <conference xmlns="jabber:config:conference">
      <public/>

      <vCard>
  <FN>Public Chatrooms</FN>
  <DESC>This service is for public chatrooms.</DESC>
      </vCard>

      <history>50</history>

      <logdir>/usr/local/var/jabberd/log/muc/</logdir>

      <sadmin>
  <user>admin@somedomain.com</user>
      </sadmin>

      <notice>
  <join>has become available</join>
  <leave>has left</leave>
  <rename>is now known as</rename>
      </notice>

    </conference>

  </service>

  <log id="muclog">
    <file>/usr/local/var/jabberd/log/muc.log</file>
    <host/>
```

```
    <logtype/>
    <format>%d: [%t] (%h): %s</format>
  </log>

  <xdb id="xdb">
    <host>conference.somedomain.com</host>

    <load>
      <xdb_file>/usr/local/jabber/xdb_file/xdb_file.so</xdb_file>
    </load>

    <xdb_file xmlns="jabber:config:xdb_file">
      <spool>/usr/local/var/spool/jabber</spool>
    </xdb_file>
  </xdb>

</jabber>
```

C *Checkpoint: Test Your Legacy Component*

Once your legacy component is set up, you should start up the component by running it in a Jabberd 1.4 process using a command similar to this:

```
/usr/local/jabber/jabberd/jabberd -c /etc/jabber/muc.xml
```

The `-c` parameter specifies a configuration file for Jabberd 1.4. After you restart your Jabberd 2 server, your new (legacy) service should be running. In the case of MU Conference, you should be able to set up multi-user chat rooms via a compatible client, or via the script included with the tarball.

5.9. Using JCR to Connect to Jabberd 2 Components

Given the newness of the Jabberd 2 server, there are few components designed for use specifically with the Jabberd 2 server. Nevertheless, Paul Curtis has written a component wrapper called the Jabber Component Runtime that is designed to run legacy components as Jabberd 2 services. The Jabber Runtime Component (JCR) allows C language components for Jabberd 1.4 to be run as standalone processes that connect to Jabberd 2. At the time of writing, only MU Conference and the Yahoo! transport have been tested with JCR. See Appendix 11 for a primer on how JCR works with Jabberd 2.

The JCR wrapper produces a binary file that runs the transport or service. Essentially, this binary file provides all the services that a separate Jabberd 1.4x process would provide for a transport or service. The INSTALL file of the JCR package provides instructions for use, and good instructions can also be found on the MU Conference site.

The following is an example of the steps that one would take to create a JCR binary for the MU Conference service:

- Download JCR and MU Conference
- Build the JCR Library
- Build the Component
- Configure the Component
- Start the Component

At the time of writing, JCR 0.2.4 and MU Conference 0.6.0 are the latest versions and are used in this example.

5.9.1. Download JCR and MU Conference

Download the latest JCR package from the JCR Home Page, and the latest MU Conference package from the MU Conference Home Page into a working directory. Then, untar the JCR package:

```
tar -xzvf jcr-0.2.4.tar.gz
```

JCR has one dependency, `glib-2.0`. You should check that this is installed on your system before building the JCR library.

5.9.2. Build the JCR Library

Build the JCR library files after entering the JCR directory:

```
cd jcr-0.2.4
make
```

I and others have noted that the build process produces errors starting with the word `TERROR_`. These errors did not prevent me from successfully using JCR.

5.9.3. Build the Component

Move the service package, in this case `mu-conference-0.6.0.tar.gz`, into the JCR directory, in this case `jcr-0.2.4`, and then untar the service package there:

```
mv ../mu-conference-0.6.0.tar.gz ./
tar -xzvf mu-conference-0.6.0.tar.gz
```

Copy the JCR library files (`main.c` and `jcomp.mk`) to the `src` directory of the service:

```
cp src/main.c mu-conference-0.6.0/src/
cp src/jcomp.mk mu-conference-0.6.0/src/
```

Then `cd` to the `src` directory of the service and build the JCR component:

```
cd mu-conference-0.6.0/src/
make -f jcomp.mk
```

The build process should create a binary (in the `mu-conference/src/` directory) called `mu-conference`. This binary provides the MU Conference service for Jabberd 2. Copy this binary to a bin directory, such as `/usr/local/bin`.

5.9.4. Configure the Component

The `src` directory of the JCR package contains an sample configuration file (`muc_conf.xml`) for use with MU Conference. Copy this file to the directory where your Jabberd 2 configuration files exist, such as `/etc/jabberd` or `/usr/local/etc/jabberd`.

Edit `muc_conf.xml` for your own installation. Within `muc_conf.xml`, set the directories for `spool`, `logdir` (there are *2* log directories) and `pidfile`. These should be set to directories over which your jabber user has permissions. (Note that `spool` refers to the spool *directory*.) Set the `host` to a name that is resolvable by your clients. Set the `ip` to the IP address or hostname of your Jabber server. Set the `secret` to match the `secret` used to identify legacy components in your `router.xml` file (found in the `local`

section). You may also wish to set an administrator JID (`sadmin`).

Edit your `router.xml` file to provide an alias for this service. Under the `Name aliases` section, add an alias that identifies this service:

```
<alias name='conference.somedomain.com' target='linker'/>
```

Where `conference.somedomain.com` is the `host` specified in `mu_conf.xml`, and `linker` is the `name` specified in `mu_conf.xml`.

5.9.5. Start the Component

Restart your Jabberd 2 server for the change to take effect, and then start the `mu-conference` binary:

```
su jabber
/usr/local/bin/mu-conference -c /etc/jabberd/muc-conf.xml
```

Your Jabber users should now be able to browse the Public Chatrooms service, and they should be able to create and use group chatrooms. The `mu-conference` binary does not detach from the terminal when started. You may wish to use Daemon Tools to start `mu-conference` on boot.

5.10. Setting up a JUD Using Users-Agent

Ryan Eatmon has updated his Users-Agent JUD (Jabber User Directory) package to work with Jabberd 2, and with some valuable help from John Hamill, I was able to get it running with my Jabberd 2 installation. Users-Agent is a Perl-based JUD that relies on MySQL for storage. Users-Agent is the JUD currently running on Jabber.org.

N *Note: JUD and vCard Data are Not the Same Thing*
A JUD provides users with a searchable database from which they can locate contacts. A JUD is *not* a database of vCard data. vCard data is separate from a JUD database, so you should not expect that upon setting up a JUD that it will contain existing vCard data. Instead, users must provide data to the JUD by registering to it.

N *Note: Fljud Provides LDAP and JUD Integration*
If you are using OpenLDAP for authentication, you may wish to look at the Fljud product. Fljud integrates a JUD with LDAP data.

I have documented the steps I followed to get Users-Agent up and running; however, I would be interested to hear how these steps work for others. A running Users-Agent package consists of three elements:

- `users-agent` (Perl script)
- `config.xml` (configuration file)
- `JUD` (MySQL database)

The `users-agent` script is the running program that provides the JUD service. The `users-agent` script relies on the `config.xml` file for configuration. The `config.xml` file must be in the same path from which `users-agent` is run or the `-c` option must be used to specify the location of `config-xml`. `users-agent` stores all data in a MySQL database named `JUD`. The package comes with a script named `createDB` that sets up the `JUD` database.

Setup of Users-Agent consists of these 7 steps, which are detailed below:

The steps may seem complex; however, installation is actually straightforward, and I have tried to provide as much detail as possible.

5.10.1. Install Required Perl Modules

Users−Agent relies on these three Perl modules:

```
Net::Jabber v1.30
XML::Stream v1.18
DBI
```

For more information about these modules, see the CPAN site. The easiest way to install these, along with any dependencies, is to use Perl's CPAN.pm install tool. To install the Perl modules, enter the `cpan` shell by running this command (as super−user):

```
perl −MCPAN −e shell
```

If this is the first time you have used the `cpan` shell, you will enter an interactive setup script. Once finished, you will be ready to install the packages above (while still in the `cpan` shell):

```
cpan> install Net::Jabber

cpan> install XML::Stream

cpan> install DBI
```

Once these are installed, you are ready to install Users−Agent. If you ever need to install additional Perl modules, you can enter the `cpan` shell by using the command above (`perl −MCPAN −e shell`).

5.10.2. Download and Install Users−Agent

Choose a directory where you would like to install Users−Agent. In the example below, User−Agent is installed under `/etc/jabberd`.

From your chosen directory, download the latest release of Users−Agent package from Jabberstudio.org. Users−Agent 1.1 is the current release and is used in the examples below.

Untar the Users−Agent package, and then switch to the directory just created:

```
tar −zxvf users−agent−1.1.tar.gz
cd users−agent−1.1
```

Users−Agent is now installed.

5.10.3. Create JUD Database

The Users−Agent package contains a `createDB` script that will create the MySQL `JUD` database. This script needs to temporary access an existing MySQL database as the means to connect to MySQL and install the `JUD` database. The easiest way to grant it access is to temporarily disable your MySQL root password by running this command:

```
mysqladmin −u root −p password ""
```

Alternately, you may edit the `createDB` script so that lines 13 and 17 contain your MySQL root password (enter it between the empty quotes). However, if you do not disable your MySQL root password, you must also export the `DBI_DRIVER` environment variable prior to running `createDB`. The `DBI_DRIVER` variable should be set to a value of `dbi:mysql:test:localhost`, where `test` is a working MySQL database.

Once you have disabled your MySQL root password (or edited `createDB` and set the `DBI_DRIVER` variable), run the script to create your `JUD` database:

```
./createDB
```

After you create the database, you should reset your MySQL root password:

```
mysqladmin −u root password "oldrootpassword"
```

Then, you can check that the database was created by entering the `mysql` console:

```
mysql −u root −p

mysql> show databases;
+----------+
| Database |
+----------+
| JUD      |
| jabberd2 |
| mysql    |
| test     |
+----------+
4 rows in set (0.00 sec)
```

The `JUD` database is now setup.

5.10.4. Set JUD Database Permissions

After the `JUD` database is created, permissions need to be granted to it. If you are using MySQL for Jabberd 2 authentication or storage, you already have a MySQL user that can be used for the `JUD` database. If this is the case, enter the MySQL console and run this SQL statement to grant the MySQL user `jabberd2` permissions to the `JUD` database (replace `jabberd2` if you are not using the default MySQL user):

```
mysql −u root −p

mysql> use JUD;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'jabberd2'@'localhost' WITH GRANT OPTION;
```

If you are using MySQL with Jabberd 2 and cannot remember the name for your MySQL user, open `sm.xml` or `c2s.xml` and scan down below `MySQL driver configuration`, where you will find the `user` and

password (`pass`) that you are using to connect with Jabberd 2.

If you are *not* using MySQL for authentication or storage, you should run a slightly different SQL statement that will create a new MySQL user and password that `users-agent` will use to connect to MySQL (replace `secret_password` with a the password you wish to use for this MySQL user):

```
mysql -u root -p

mysql> use JUD;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'jabberd2'@'localhost'
    -> IDENTIFIED BY 'secret_password' WITH GRANT OPTION;
```

Once you have granted permissions, you can test them by using your `jabberd2` user to start the MySQL console. You will need your `jabberd2` MySQL password (created immediately above or found in `sm.xml`) to start the console:

```
mysql -u jabberd2 -p

mysql> use JUD;
mysql> select * from jud;
```

You should receive output something like: `Empty set (0.01 sec)`. Your `JUD` database is now ready for use by `users-agent`.

5.10.5. Configure `config.xml`

Users-Agent relies on `config.xml` for configuration. Open `config.xml` in a text editor. To get Users-Agent running, I edited the values between these 6 tags:

```
<hostname>
<port>
<secret>
<name>
<username>
<password>
```

1. Edit the `hostname` value to provide the hostname of your Jabberd 2 server. This should match the `id` at the top of `sm.xml`.
2. Edit `port` to provide the port number that the Jabberd 2 router is listening on. By default, this would be `5347`.
3. Edit `secret` to provide the shared `secret` that the Jabberd 2 router uses to connect to legacy components. The shared `secret` you are currently using can be found in `router.xml` near the top. Look for the `secret` tag pair under the section labeled `Local Network configuration`.
4. Edit `name` to provide the hostname for the Users-Agent. This hostname is usually something like `users.somedomain.com`, `users.jabber.somedomain.com`, `jud.somedomain.com`, etc. This hostname *must* be resolvable via DNS if your clients are to access it over the Internet.
5. Edit `username` to provide the MySQL user (from step 5.10.4.). For default installations, the `username` will be `jabberd2`.
6. Edit `password` to provide the MySQL user password (from step 5.10.4.). Again, this password can be found in `sm.xml` or `c2s.xml` if you are already using MySQL with Jabberd 2.

`users-agent` is now configured for use.

5.10.6. Test `users-agent`

You should now be ready to start and test `users-agent`. Make sure that your Jabberd 2 and MySQL servers are running. Then, start `users-agent`:

```
su jabber
./users-agent
```

Users-Agent should start up without returning any errors, and it should not exit. Users-Agent does not put itself into the background, so it should remain running in the terminal.

Log into your server from a Jabber client. With `users-agent` running, you should be able to see `Users-Agent` as an available service. How your client displays this depends on the client. The `Users-Agent` service should offer both `Register` and `Search` options.

Your `JUD` database is empty, so you will need to register some users to test it. From your Jabber client, select `Register` (this is usually a right-click option in Jabber clients). You should be presented with the option of registering for this service along with a form into which you can enter personal information.

I *Important: Choice of Client Affects Use of Agent*

During testing, I was not able to get Users-Agent to perform consistently across test clients. For Linux, PSI seemed to respond the best; however, PSI does not display a registration form the first time a user attempts to register. The user must register once and then re-register to add any data. On Windows, JAJC and Exodus performed the best; however, JAJC, did not provide `register` and `search` options when right-clicking on the `Users-Agent`. Selecting `Discovery (info)` did provide these options in JAJC. Exodus performed the best of all the clients I tested (PSI, Gabber, Gnome-Gabber, Kopete, Exodus, JAJC and Neos).

Now, you should be able to actually search the Users-Agent. From your client, return to this service and select `Search`. You should be presented with a search form.

I *Important: Users-Agent Supports Partial Matching*

Users-Agent supports partial matching for searches. Wildcards are not needed and are not used.

You should be able to search for the user you just registered by searching on the first letter of any of the fields entered into the registration form.

5.10.7. Set Users-Agent to Start with Jabberd 2

Once you have Users-Agent configured and running, you may wish to add it to your RC script to start and stop it with Jabberd 2, or you may choose to use Daemon Tools to monitor it.

I *Important: User-Agent Dies if it Cannot Connect to Router*

`users-agent` will die if it cannot connect to the `router` component. This means that if your Jabberd 2 server is stopped or dies, `users-agent` will die and therefore need restarting. You may wish to use Daemon Tools to make sure that `users-agent` is always restarted if it cannot connect to the `router`.

5.11. Integrating Users−Agent with vCard Data (private servers only)

I have written a patch for Users−Agent that sets Jabberd 2 vCard data as the data source for Users−Agent. Once applied, Users−Agent uses the Jabberd vCard table for searches and registration. Thus, user updates to their own vCards are reflected in Users−Agent JUD searches. Users are able to update their vCard data in the normal way, or they can update the fields captured by Users−Agent (first name, last name, nickname and email) through the Users−Agent registration procedure.

I *Important: JUD and vCard Data SHOULD NOT Be Integrated on Public Servers*

This patch is intended for *private Jabberd 2 servers only*. On public servers, vCards represent personal information over which the user has complete control. A user is able to allow another user to view personal vCard data by either authorizing that user, or by manually registering to the JUD.

This patch *is* intended for private Jabberd 2 servers, such as may be used by corporations or education institutions. For these setups, it is often convenient for each user to be able to search a list of all users. Note, however, that even after this patch is applied, each user must create a vCard entry to become visible in the Users−Agent JUD.

I *Important: Patch Not Tested Under Heavy Load*

This patch has not been tested under heavy load.

In order to apply the patch, first get Users−Agent running as in Section 5.10. Then, download the users−agent.vcard.patch into the directory where your `users−agent` file is installed. Apply the patch with this command:

```
patch −p0 < users−agent.vcard.patch
```

Edit `config.xml` so that `dbname` is the name of your Jabberd 2 MySQL database (usually `jabberd2`). Edit the `username` and `password` if necessary. Restart `users−agent`, and it should now be connecting to Jabberd 2 vCard data.

Back	Up	Next

6. Common Administrative Tasks

This section attempts to provide a guide for the most common day−to−day Jabberd administrative tasks, including user management.

- Starting and Stopping Jabberd 2
- Converting from Jabberd 1.4
- Adding Users
- Removing Users
- Sending MOTD's and Messages to All Online Users

Detailed configuration information begins in the next section.

6.1. Starting and Stopping Jabberd 2

This section describes how to start and stop the Jabberd 2 binaries. See Appendix 7 for running Jabberd 2 with an RC script or Appendix 8 for running Jabberd 2 using Daemontools.

6.1.1. Starting Jabberd 2

The `jabberd` start script starts the Jabberd 2 server. This script is installed in the same directory with the Jabberd 2 binaries (`/usr/local/jabberd` by default). To start the server, switch to the `jabber` user and run the start script:

```
su jabber
/usr/local/bin/jabberd
```

The script handles the starting of the five Jabberd 2 binaries. The script assigns the relevant configuration file (located in `/usr/local/etc/jabberd` by default) for each Jabberd 2 binary.

6.1.2. Stopping Jabberd 2

There is currently no shutdown script for Jabberd 2; therefore, the five Jabberd 2 processes must be killed manually in order to shut the server down. To stop the server, kill the following processes (either as root or the jabber user):

```
router
resolver
sm
c2s
s2s
```

Stopping one of the processes above may cause another to die; however, *it is important to make sure that all Jabberd 2 proccesses have died before attempting a restart.* A single running process may prevent the server from restarting.

6.2. Converting from Jabberd 1.4

Robert Norris, the primary Jabberd developer, has created a Perl script that migrates Jabberd 1.4 data to either MySQL or PostgreSQL. Specifically, the script migrates only authentication information and rosters. The migrate.pl script is currently available from CVS on jabberstudio.org; however, it will be released with the next Jabberd 2 release.

Essentially, the script works as a service that connects to an existing Jabberd 1.4 installation, and it exports to an existing Jabberd 2 database. It has several dependencies (as listed in the script):

- XML::Stream 1.17 or higher (from JabberStudio)
- Net::Jabber 1.29 or higher (from JabberStudio)
- Digest::SHA1
- DBI
- DBD::Pg or DBD::mysql

The migrate.pl file contains complete instructions for use.

6.3. Adding Users

At the time of writing, there are no scripts for adding users; however, users can be added easily to a MySQL or PostgreSQL database. This section describes how to add a user to a MySQL database. Adding a user to PostgreSQL, would be similar. This section assumes that public registration has been disabled (see Section 5.5).

***N** Note*

Jabberd 2 does not provide a facility for adding users to− or removing users from a Berkeley DB authentication store.

6.3.1. Enabling Auto−Create

The `user` section of `sm.xml` contains a tag named `auto−create`, and enabling this tag causes a user to be created in the storage package the first time a user logs on if that user does not yet exist in that database. With this tag enabled, users can be manually added to the authentication (`authreg`) database, and when each user logs in for the first time, Jabberd will create rosters, templates, etc. for that user. Specifically, the `auto−create` tag causes the `user−create` chain to be called when a new user logs in for the first time.

Auto−create is disabled by default. To enable it, uncomment the `auto−create` tag in the `user` section of the `sm.xml` file. This tag is near the end of the `sm.xml` file.

6.3.2. Creating Users from MySQL Console

Now that `auto−create` is enabled, you can create a user from the MySQL console.

Log onto the MySQL console as the Jabberd user:

```
mysql −u jabberd2 −p
```

From the MySQL console, Switch to the `jabberd2` database:

```
mysql> use jabberd2
```

From the MySQL console, Insert a row into the `authreg` table. The row should contain values for `username`, `realm` and `password` :

```
mysql> insert into authreg (username, realm, password)
    −> values ('myusername', 'somedomain.com', 'mypassword');
```

Change the value of `somedomain.com` to match your configuration. In most cases, the realm will be the same as the ID of the Jabber server. Change `myusername` and `mypassword` to the name and password for your new user.

Your new user will now be able to log on using a Jabber client.

6.4. Removing Users

The MySQL console can be used to delete users from a MySQL authentication database (authreg). Enter this command to delete a user:

```
mysql> delete from authreg where username='the_user_name';
```

Note that deleting the user from the `authreg` table will not delete additional user data, such as rosters; however, once the user has been deleted from the `authreg` table, the user will no longer be able to log on to the Jabberd server.

6.5. Sending MOTD's and Messages to All Online Users

Sending MOTD's (Messages of the Day) and messages to all online users must be done from an administrative account with `broadcast` privileges. See Section 5.4 for information about how to assign these privileges. Once an account has `broadcast` privileges, these messages can be sent from a Jabber client. Add the following contacts to the administrative account you have created:

```
somedomain.com/announce

somedomain.com/announce/online
```

Sending a message to `somedomain.com/announce` will send a Jabber MOTD to all users on the domain. Offline users will receive the message the next time they log on. Sending a message to `somedomain.com/announce/online` sends a message only to online users. Note that if your Jabber client does not provide explicit support for MOTD's, you can add a contact with the JID's above, and use that contact to send MOTD's.

Back	Up	Next

7. Router.xml Configuration

The `router.xml` file provides configuration for the *router* component, which is the backbone of the Jabberd server. The router component accepts connections from other components, and it passes XML packets between them. The `router.xml` file contains these primary sections:

- ID, PID and Logging
- Network
- Input/Output Control
- Aliases
- Feature access controls

As readers are probably aware, the Jabberd XML files are self−documenting. This and subsequent configuration sections attempt merely to provide an overview of the structure of these files in addition to tips not found in the files themselves.

7.1. ID, PID and Logging

The first three subsections of `router.xml` handle basic housekeeping items. For nearly all installations, the `id` section should be kept as default (`router`). The `pid` section specifies where the *router* component should write its process ID. If you do not need to be able to read the process ID from outside of Jabberd, you may comment this section out. For example, if you were to use D.J. Bernstein's DaemonTools to run Jabberd, you would comment the `pid` section.

Jabberd logging defaults to the `syslog`. If you prefer the `router` to write to its own log file, change the `log type` to `file`, and specify a location for the log.

7.2. Network

The `local` subsection handles network and connection settings for the *router*. The `ip` and `port` tags specify the network settings.

***N** Note: `0.0.0.0` Specifies All Available IP's*

Note that in all of the Jabberd XML files, an IP address specified as `0.0.0.0` specifies that the component should listen on all available IP addresses. `0.0.0.0` is the default setting for IP addresses, and keeping this default setting should be fine for most installations.

The `users` tag specifies the file containing user ID's and passwords for users that can access the *router*. See the next section for a description of this file.

The `secret` tags specify a shared secret for legacy services. Legacy services are authenticated via this shared secret, as opposed to being authenticated against the user list in the `router-users.xml` file. This *secret* is tantamount to the shared secret specified for Jabberd 1.4 linked configuration files. See Section 5.8 and 5.9 for information on setting up legacy services.

The `pemfile` location specifies the location of the certificate and private key for client communications. Note that in this instance, *client* refers to clients of the *router*, and not end-user Jabber clients. That is to say, the *pemfile* specified here is used to secure communications between other Jabberd components and the *router* itself. See Section 5.2 and Appendix: Generating A Self-Signed SSL Key for more information about setting up SSL on Jabberd. Commenting this section has the effect of disabling SSL communication between Jabberd components.

7.3. Input/Output Control

The `i/o` section controls the following input/output options:

- Connection Limiting
- Rate Limiting
- IP Access Control

Note that the default settings for these subsections should be fine for most installations.

7.3.1. Connection Limiting

Jabberd sets a limit on the number of connections via the `max_fds` (maximum file descriptors) setting. Jabberd uses a file descriptor for each connection. Thus, setting a maximum number of file descriptors for the router has the effect of limiting the number of concurrent connections for the Jabberd server. As the `router.xml` file notes, the *router* itself can use up to 4 connections (with other Jabberd components); therefore, the maximum number of file descriptors set here is actually slightly greater than the number of potential concurrent connections.

7.3.2. Rate Limiting

The `limits` subsection dictates throttling for individual connections. This section is tantamount to a simplified *karma* setup as found in Jabberd 1.4. The default for both `bytes` and `connects` is `0`. Thus, these limits are disabled by default. Administrators of servers under heavy load may wish to set limits here to prevent users from controlling excessive server resources. The `router.xml` file contains examples for setting rate limiting on connections.

7.3.3. IP Access Control

The `access` subsection specifies IP addresses that should be allowed or denied access to the *router*. IP addresses denied access to the *router* cannot have their packets handled and are thus denied access to Jabberd server functions.

|| QUESTION: Is the above statement actually correct? ||

The `order` subsection specifies the order of rule checking (checking of allow rules then deny rules versus checking of deny rules then allow rules). IP restrictions are set using either an `allow` or a `deny` tag below the `order` within the `access` subsection. Omission of both a deny and allow rule causes all connections to be accepted as is the default setting.

7.4. Aliases

The `aliases` section provides support for linked legacy services, such as MU Conference and JUD. These legacy services connect directly to the *router* via linked configuration. Each `alias` specifies the resolvable `name` for the component and the `target` specifies the `service id` as specified in the linked component XML file. See Section 6.5 for more information about linking legacy services.

7.5. Feature Access Controls

The `aci` section specifies the type of access for each user specified in `router-users.xml`. By default, the `jabberd` user is granted all access because the other Jabberd components connect to the *router* as the `Jabberd` user. Of special note in this section is the example of a user with the 'acl type = 'log''. Such a user would be able to receive all packets that pass through the router, and thus, all packets handled by the Jabberd server.

Although Jabberd does not have a facility for logging all packets, such a facility could be easily provided by specifying a user with 'acl = 'log'' permissions and creating a component that would connect to the router as this user. Such a component would then be able to log all Jabberd packets.

| Back | Up | Next |

8. Router-users.xml Configuration

The sole purpose of the `router-users.xml` file is to provide a list of user and password pair(s) for authentication by the *router*. The default user (`jabberd`) is the user against which the other Jabberd components (*sm*, *resolver*, *s2s* and *c2s*) authenticate.

The `router-users.xml` file probably does not deserve its own section, ***except to encourage strongly that all Jabberd administrators change the default password on production servers***. Using the default password (`secret`) leaves your Jabberd installation open to easy attack by malicious crackers.

Once the default password in `router-users.xml` is changed, the corresponding passwords must be changed in the following files:

```
sm.xml
resolver.xml
s2s.xml
```

```
c2s.xml
```

These passwords are found in the sections labeled `Router connection configuration` near the top of the above listed files.

| Back | Up | Next |

9. Sm.xml Configuration

The `sm.xml` file configures the *session manager* component of Jabberd 2. The *session manager* acts as a layer between the router and the externally available components s2s and c2s. The `sm.xml` file configures the following functions:

- Jabberd Identification
- Communication with the Router
- Logging
- Database Connection and Configuration
- Access Control for Administrative Functions
- Modules that Are Called during Sessions
- Static Discovery Settings for Legacy Components
- User Options

Below is an overview of the settings in the `sm.xml` file.

9.1. Jabberd Identification

The first two sections of `sm.xml` specify the ID on the network and the PID file to write. The `id` section is important because it is this ID that becomes appended to user names to form a JID (Jabber Identification). For most installations, `id` will be the hostname something like `somedomain.com`. This ID must be resolvable via DNS if the server is to communicate with other Jabber servers; therefore, it need not be resolvable for closed or local Jabberd systems. An IP address may be used as the 'id'; however, because an IP address is not resolvable via DNS, a system based on an IP address ID would not be able to communicate with other Jabber servers.

The `pid` section specifies the location of the PID file. This may be commented out if a PID file is not needed.

9.2. Communication with the Router

The `router` section controls communication with the *router* component. The default `ip` and `port` should be fine for most installations, although note that if the *session manager* is running on a separate server, an external IP address would be specified here.

The `user` and `pass` sub−sections specify the user name for connecting to the *router*. These must match against a pair specified in `router−users.xml` as explained in Section 9. Basic security procedures dictate that the default password should be changed for production systems.

The `pemfile` section specifies the certificate and private key to be used for communication with the *router*. See Section 5.2 and Appendix: Generating A Self−Signed SSL Key for more information about setting up SSL on Jabberd. Commenting this section has the effect of disabling SSL communication between the *session*

manager and *router*.

The `retry` section specifies how the *session manager* should try to reconnect to the *router* if the connection cannot be established during startup or if the connection is lost during operation. The default settings prevent the *session manager* from indefinitely attempting to reconnect if this connection cannot be made. These default settings will essentially cause the *session manager* to die if the router dies or is killed.

9.3. Logging

Jabberd logging defaults to the `syslog`. If you prefer the *session manager* to write its own log file, change the `log type` to `file`, and specify a location for the log.

9.4. Database Connection and Configuration

The *session manager* handles the database connection referred to in this guide as the *application data package* (as opposed to the *authentication data package*). You need to edit only the sub−section applicable to your choice of *application data package* (MySQL, PostgreSQL or Berkeley DB). For database packages running on the same server as the *session manager* you should need to edit only the password used to connect to the database (this is not necessary for Berkeley DB). Change the `host` for databases running on separate machines. Note that the default password (`secret`) should not be used on production machines.

9.5. Access Control for Administrative Functions

The `acl` section controls which users have access to administrative functions. Among these functions are broadcast, messages and discovery. Note that specification of an administrative user in this section does not cause that user to be created in the authentication (*authreg*) database. Thus, you must register a user separately. Section 7.4 describes how to use an administrative account to send messages to all users.

Administrative functions currently have only minimal support in Jabber clients; however, the JAJC client for Windows supports the administration functions discovery of online users and messages to online users.

9.6. Modules that Are Called during Sessions

The `modules` section controls module chains that are called by specific events. Events are defined in terms of receipt or delivery of a type of packet. These module chains should be edited only by experienced Jabberd administrators. Of note are the chains that are called during user creation (`user−create`) and user deletion (`user−delete`).

9.7. Static Discovery Settings for Legacy Components

The `discovery` section contains both standard service information for instant messaging (the `identity` sub−section) in addition to settings for static discovery settings for legacy components (the `items` sub−section). If your server is running legacy components, such as JUD, MU Conferencing, MSN−T or Aimtrans, you should complete an `item` sub−section if the component does not support *disco* (discovery). Follow the examples shown in this section to add an item for static discovery for your legacy components.

9.8. User Options

The `user` section specifies templates that should be added to a user's data store when the user is created, and this section also specifies whether non−existent user should be created when that user logs in. This option allows an administrator to add users to the authentication database (*authreg*) and then have those users be

created the first time they log in.

Back | Up | Next

10. Resolver.xml Configuration

The `resolver.xml` file configures the router component which handles hostname resolution for the *s2s* (server to server) component. The `resolver.xml` file configures the following functions:

- PID File
- Communication with the Router
- Logging

Configuration of the *resolver* is concerned primarily with how the *resolver* and *router* components communicate.

10.1. PID File

The `pid` section specifies the location of the PID file. This section may be commented if a PID file is not needed.

10.2. Communication with the Router

The `router` section controls communication with the *router* component. The default `ip` and `port` should be fine for most installations, although note that if the *resolver* is running on a separate server, an external IP address would be specified here.

The `user` and `pass` sub−sections specify the user name for connecting to the *router*. These must match against a pair specified in `router−users.xml` as explained in Section 9. Basic security procedures dictate that the default password should be changed for production systems.

The `pemfile` section specifies the certificate and private key to be used for communication with the *router*. See Section 5.2 and Appendix: Generating A Self−Signed SSL Key for more information about setting up SSL on Jabberd. Commenting this section has the effect of disabling SSL communication between the *resolver* and *router*.

The `retry` section specifies how the *resolver* should try to reconnect to the *router* if the connection cannot be established during startup or if the connection is lost during operation. The default settings prevent the *resolver* from indefinitely attempting to reconnect if this connection cannot be made. These default settings will essentially cause the *resolver* to die if the router dies or is killed.

10.3. Logging

Jabberd logging defaults to the `syslog`. If you prefer the *resolver* to write its own log file, change the `log type` to `file`, and specify a location for the log.

Back | Up | Next

11. S2s.xml Configuration

The `s2s.xml` file handles configuration for the server−to−server Jabberd component. The `s2s.xml` file provides network settings for this component in addition to settings for communicating with the router component:

Below is an overview for the settings in the `s2s.xml` file.

11.1. PID File

The `pid` section specifies the location of the PID file. This section may be commented if a PID file is not needed.

11.2. Communication with the Router

The `router` section controls communication with the *router* component. The default `ip` and `port` should be fine for most installations, although note that if *s2s* is running on a separate server, an external IP address would be specified here.

The `user` and `pass` sub−sections specify the user name for connecting to the *router*. These must match against a pair specified in `router−users.xml` as explained in Section 9. Basic security procedures dictate that the default password should be changed for production systems.

The `pemfile` section specifies the certificate and private key to be used for communication with the *router*. See Section 5.2 and Appendix: Generating A Self−Signed SSL Key for more information about setting up SSL on Jabberd. Commenting this section has the effect of disabling SSL communication between *s2s* and *router*.

The `retry` section specifies how *s2s* should try to reconnect to the *router* if the connection cannot be established during startup or if the connection is lost during operation. The default settings prevent *s2s* from indefinitely attempting to reconnect if this connection cannot be made. These default settings will essentially cause *s2s* to die if the router dies or is killed.

11.3. Logging

Jabberd logging defaults to the `syslog`. If you prefer *s2s* to write its own log file, change the `log type` to `file`, and specify a location for the log.

11.4. Network Configuration

The `local` section specifies network configuration for *s2s* in addition to the secret used for dialback keys. The default IP address and port should be fine for most installations (the 0.0.0.0 setting allows *s2s* to listen on all available IP addresses). The `secret` setting specifies the passphrase that *s2s* uses to generate dialback keys for other Jabber servers. The default setting of `secret` should be changed on production servers.

11.5. S2S Connection Checking

The `check` section handles checking of connections with other Jabber servers. By default, these checks are disabled (`interval` is set to `0`). To enable checking, set an `interval` in seconds, and then set intervals for queue expiry, invalid route expiry and/or keep alives.

|| QUESTION: Under what conditions would connection checking be useful? ||

| Back | Up | Next |

12. c2s.xml Configuration

The `c2s.xml` file configures the client−to−server Jabberd component. The *c2s* component handles communications with Jabber clients, and the settings in `c2s.xml` are primarily concerned with client communication:

- PID File
- Communication with the Router
- Logging
- Network Configuration
- Input/Output Control
- Client Authentication and Registration
- Authentication Encryption

Below is an overview of the settings in the `c2s.xml` file.

12.1. PID File

The `pid` section specifies the location of the PID file. This section may be commented if a PID file is not needed.

12.2. Communication with the Router

The `router` section controls communication with the *router* component. The default `ip` and `port` should be fine for most installations, although note that if *c2s* is running on a separate server, an external IP address would be specified here.

The `user` and `pass` sub−sections specify the user name for connecting to the *router*. These must match against a pair specified in `router−users.xml` as explained in Section 9. Basic security procedures dictate that the default password should be changed for production systems.

The `pemfile` section specifies the certificate and private key to be used for communication with the *router*. See Section 6.3 and Appendix: Generating A Self−Signed SSL Key for more information about setting up SSL on Jabberd. Commenting this section has the effect of disabling SSL communication between *c2s* and *router*.

The `retry` section specifies how *c2s* should try to reconnect to the *router* if the connection cannot be established during startup or if the connection is lost during operation. The default settings prevent *c2s* from indefinitely attempting to reconnect if this connection cannot be made. These default settings will essentially

cause *c2s* to die if the router dies or is killed.

12.3. Logging

Jabberd logging defaults to the `syslog`. If you prefer *c2s* to write its own log file, change the `log type` to `file`, and specify a location for the log.

12.4. Network Configuration

The `local` section specifies network configuration for *c2s*. For most installations, the `id` should be the same as the ID specified in `sm.xml`. The `realm` attribute can be used in cases where multiple jabber servers are relying on the same authentication database. This might be the case for a large company with several Jabber servers that authenticate via LDAP. The default IP address and port should be fine for most installations (the 0.0.0.0 setting allows *c2s* to listen on all available IP addresses).

The `pemfile` sub−section of the `local` section specifies the certificate and private key to be used for *client* communications. See Section 5.2 and Appendix: Generating A Self−Signed SSL Key for more information about setting up SSL on Jabberd. Commenting this section has the effect of disabling SSL communication between Jabberd and clients. Note that this key pair does not need to be the same key pair used for internal Jabberd communications.

12.5. Input/Output Control

The `i/o` section controls the following input/output options:

- Connection Limiting
- Rate Limiting
- IP Access Control

Note that the default settings for these subsections should be fine for most installations.

12.5.1. Connection Limiting

Jabberd sets a limit on the number of connections via the `max_fds` (maximum file descriptors) setting. Jabberd uses a file descriptor for each connection. Thus, setting a maximum number of file descriptors for *c2s* has the effect of limiting the number of concurrent *client* connections for the Jabberd server. As the `c2s.xml` file notes, *c2s* itself can use up to 5 connections (with other Jabberd components); therefore, the maximum number of file descriptors set here is actually slightly greater than the number of potential concurrent connections.

12.5.2. Rate Limiting

The `limits` subsection dictates throttling for individual connections. This section is tantamount to a simplified *karma* setup as found in Jabberd 1.4. The default for both `bytes` and `connects` is `0`. Thus, these limits are disabled by default. Administrators of servers under heavy load may wish to set limits here to prevent users from controlling excessive server resources. The `c2s.xml` file contains examples for setting rate limiting on connections.

12.5.3. IP Access Control

The `access` subsection specifies IP addresses that should be allowed or denied access to *c2s*. IP addresses denied access to *c2s* cannot have their packets handled and are thus denied access to Jabberd server functions.

The `order` subsection specifies the order of rule checking (checking of allow rules then deny rules versus checking of deny rules then allow rules). IP restrictions are set using either an `allow` or a `deny` tag below the `order` within the `access` subsection. Omission of both a deny and allow rule causes all connections to be accepted as is the default setting.

12.5.4. Client Connection Checking

The `check` section handles checking of connections with other Jabber clients. By default, these checks are disabled (`interval` is set to `0`). To enable checking, set an `interval` in seconds, and then set intervals for queue expiry, invalid route expiry and/or keep alives.

12.6. Client Authentication and Registration

The `authreg` section controls aspects of client authentication and registration:

- Authentication Package
- Public Registration
- Password Change
- Authentication Mechanisms
- Authentication Package Configuration

These sub−sections are described below.

12.6.1. Authentication Package

The `module` subsection specifies the authentication data package to use. Jabberd sets this `module` during build according to your configuration choice, and so you should not need to edit this unless you change the authentication data package after installing Jabberd.

12.6.2. Public Registration

The `enable` subsection of `register` controls whether registration is publicly open for new users. Public registration is enabled by default. Comment the `enable` tag to disable public registration.

12.6.3. Password Change

The `password` subsection of `register` specifies whether users can change their own passwords. Password change is disabled by default. As `c2s.xml` notes, it may be useful to enable password change if public user registration is disabled.

12.6.4. Authentication Mechanisms

The `mechanisms` sub−section specifies which authentication encryption methods will be offered to clients. As `c2s.xml` notes, you should comment out any mechanisms that you do not want to offer.

12.6.5. Authentication Package Configuration

The last sub−section of `authreg` controls connectivity with your authentication data package. The default settings should be fine for most installations unless your authentication data package is running on a separate server.

| Back | Up | Next |

13. Jabberd 2 Architecture (Draft)

***N** Note: This Section Is Work in Progress*

Note that this section is largely a work in process. Thus, information below should be regarded as draft documentation, and *not* as a finalized description of Jabberd 2 architecture. Essentially, I am adding to this section as I learn about Jabberd 2.

|| TODO: Add architecture introduction consisting of architecture feature list −or− just new feature list ||

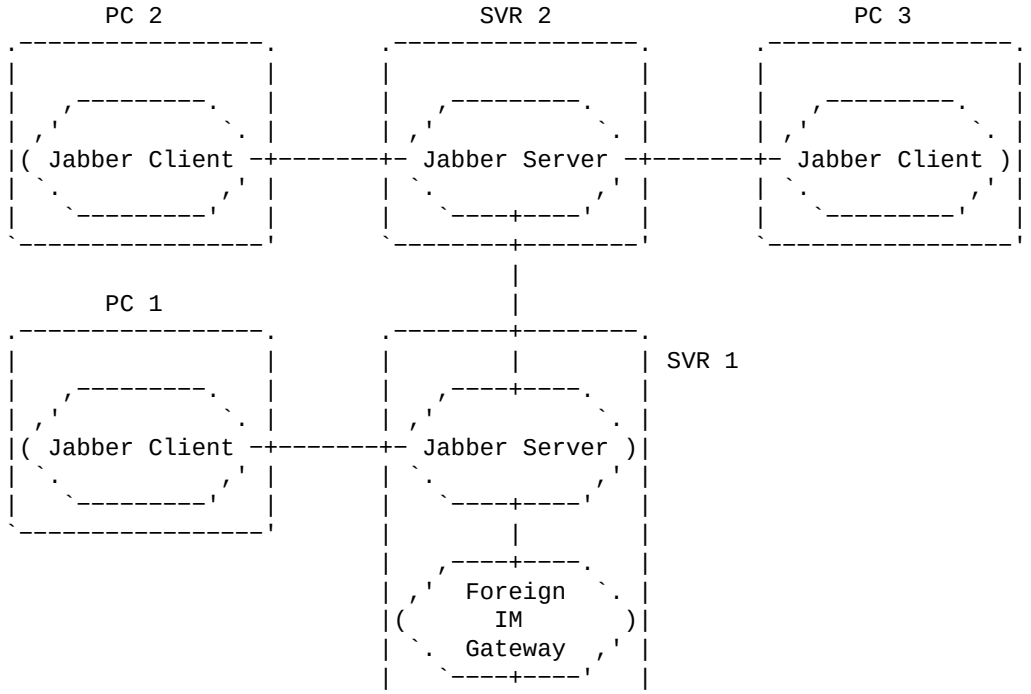
13.1. Jabber Network Architecture

Jabber, in the form of the XMPP protocol, provides a protocol for messaging, and as such it provides a standardized platform for Jabber server communication:

- Jabber Client to Jabber Server
- Jabber Server to Jabber Server
- Jabber Server to Foreign IM Gateway

Figures 13.1 and 13.2 below demonstrate how these three types of communication can occur.

***Figure 13.1.1. Jabber Deployment Diagram (High Level View)**:*



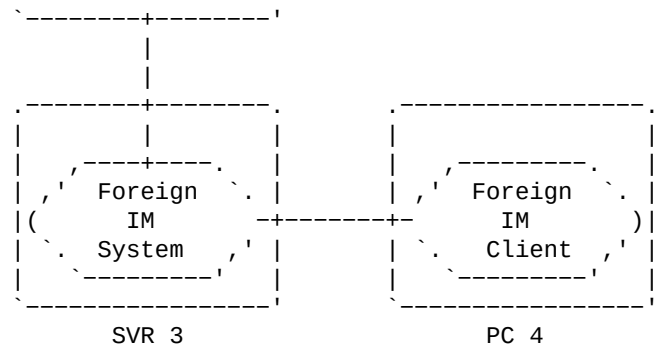
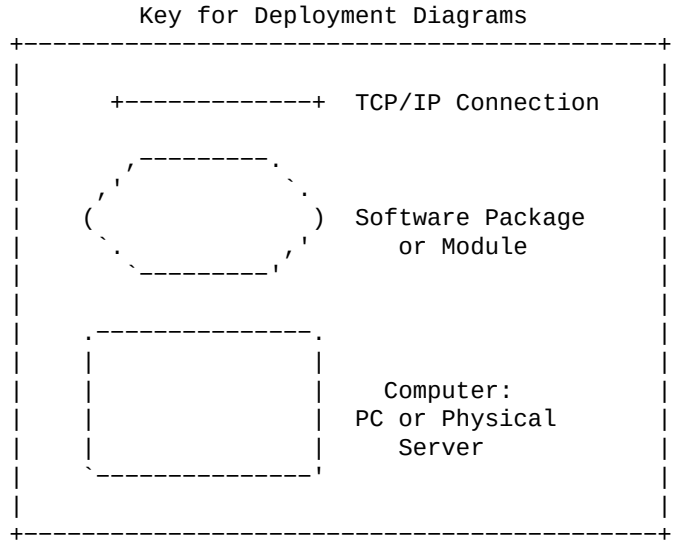


Figure 13.1. Adapted from XMPP Core Draft

Figure 13.1.2. Key to Deployment Diagrams:



13.1.1. Jabber Client to Jabber Server

In the diagram above, Jabber Clients on *PC 2* and *PC 3* are able to communicate via Jabber IM provided by *SVR 2*. Both clients have accounts on this server, and this part of the diagram, if taken by itself, would represent a *closed* or *private* Jabber system.

Note that a *Jabber Client* is not necessarily a user−based IM client. For example, the client running on *PC 3* might actually be part of a web server. In this manner, a user on *PC 1* might make updates to the web server by using a Jabber client.

13.1.2. Jabber Server to Jabber Server

Communication between *SVR 1* and *SVR 2* demonstrates how Jabber employs a distributed architecture. Clients on *PC 1* and *PC 2* are able to communicate with each other even though these clients have accounts on separate servers. These clients do not need to know anything about the remote servers. Instead, each client needs only to know the address of the client with which it wishes to communicate.

Jabber servers running on *SVR 1* and *SVR 2* rely on the Domain Name Service (DNS) for address lookup in order to communicate with each other. In this manner, the Jabber IM system resembles the email network architecture provided by POP and SMTP, the most widely used email protocols on the Internet. The ability to

resolve names of other jabber servers and to route messaging makes jabber unlike proprietary IM systems. Jabber does not rely on a centralized server farm. Thus, Jabber is easily scalable, and it can be used as a closed or open system.

13.1.2. Jabber Server to Foreign IM Gateway

In this diagram, *SVR 1* communicates with a *Foreign IM Gateway* running on the same machine. This gateway is able to communicate with a *Foreign IM System*, such as AOL, MSN, Yahoo or IRC. Connection with this communication gateway allows clients on *PC 1* and *PC 4* to communicate despite the fact that the user on *PC 1* is running Jabber client software, while the user on *PC 4* is running software, such as AIM, that uses a completely different messaging protocol.

The *foreign IM* example demonstrates the flexibility that Jabber provides. The XMPP protocols dictate set of XML−based communication standards. Therefore, a gateway for other protocols can be created, provided that the details of such a non−XMPP protocol are known.

13.2. Jabberd 2 Component Architecture

***N** Note: Subsequent Documentation is Jabberd 2 Specific*

> The preceding section provides an overview of how the Jabber architecture works. Other Jabber servers, in addition to Jabberd 2, provide the same services. However, the documentation in the subsequent sections deals with Jabberd 2 specifically.

The beauty of the Jabberd 2 architecture lies in the fact that its component architecture distributes services across five components, each of which communicates over TCP/IP:

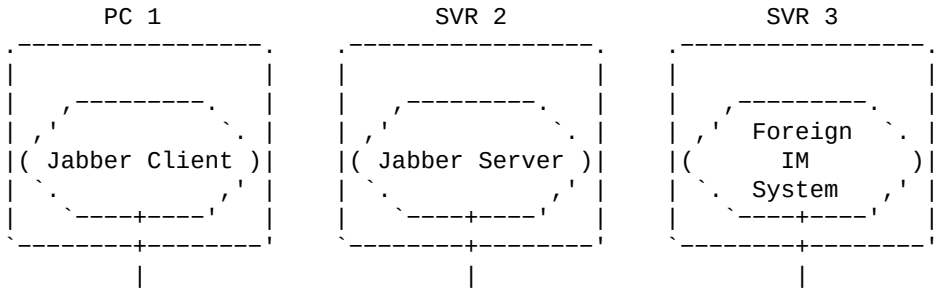
- Router
- Server to Server (S2S)
- Resolver
- Session Manager (SM)
- Client to Server (C2S)

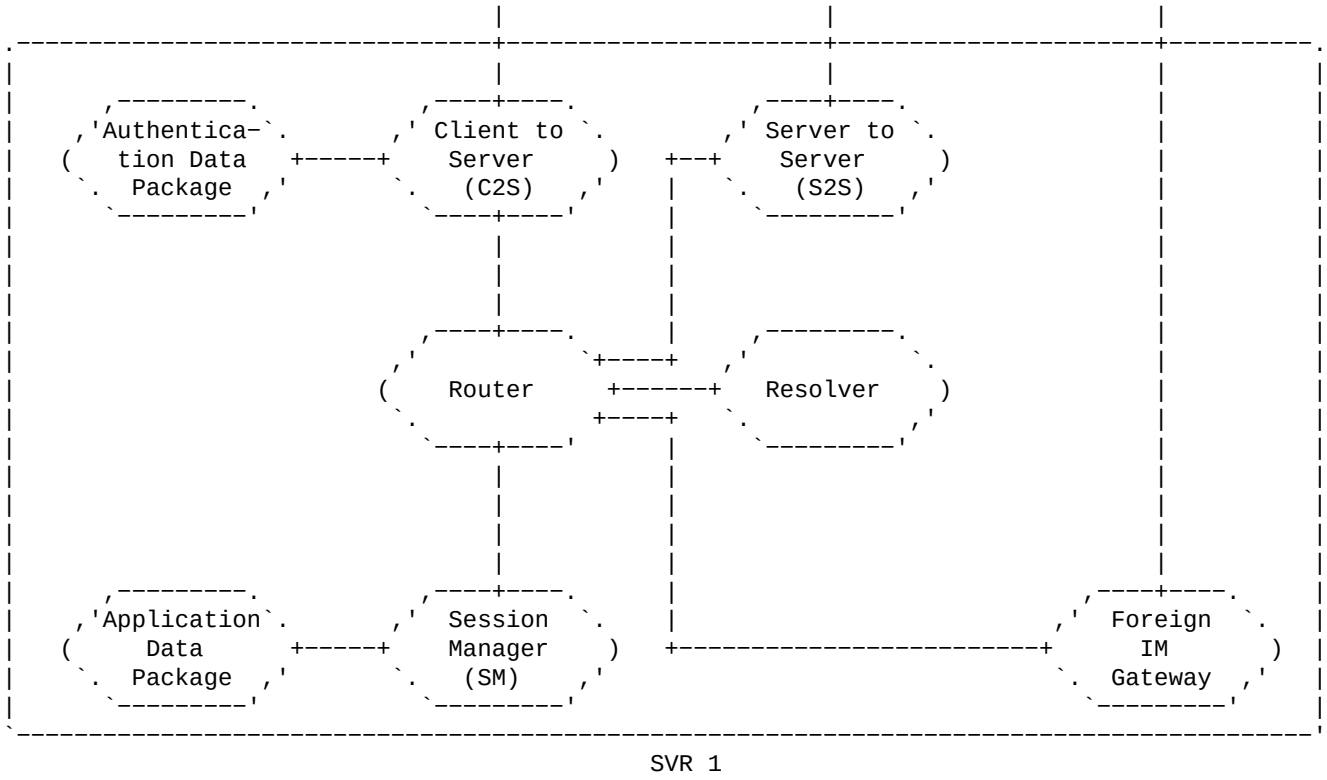
Jabberd 2 also relies on third−party components:

- Application Data Store
- Authentication Data Store
- Foreign IM Gateway(s)

Figure 13.2 shows how these components communicate and how they are generally located on a single physical server.

Figure 13.2. Jabberd Deployment Diagram (Component Level View):





***N** Note: Jabberd 2 Relies on External Packages*

The diagram above shows all the components required for a functioning Jabberd 2 server in addition to one optional component, a *Foreign IM Gateway*. Foreign IM Gateways are not included with the Jabberd 2 distribution. Packages for authentication and application data are also not included with the Jabberd 2 Distribution. Instead, Jabberd 2 connects to third−party data and authentication packages that must be installed in order for Jabberd to function.

Figure 13.2.1 shows how Jabberd 2 distributes functions across five separate modules. Just as the data stores and gateway(s) may be located on separate machines, the Jabberd 2 components may be easily distributed across machines because each component consists of a configuration file and a binary executable that communicates via TCP/IP. This architecture allows the server to scale. When load on a Jabberd 2 server becomes high, component(s) can be moved to separate machines.

13.2.1. Router

The *Router* is the backbone of the Jabberd server. It accepts connections from Jabberd components and passes XML packets between components.

13.2.2. S2S

The *S2S* (Server to Server) component handles communications with external servers. *S2S* passes packets between other components and external servers, and it performs dialback to authenticate remote Jabber servers.

13.2.3. Resolver

The *Resolver* acts in support of the *S2S* component. The *Resolver* resolves hostnames for *S2S* as part of dialback authentication.

13.2.4. SM

The *SM* (Session Manager) component implements the bulk of the instant messaging features:

- Message passing
- Presence
- Rosters
- Subscriptions

The *SM* component connects to the "Application Data Package *(db) in order to provide persistant data storage. Additionally, the* SM *component handles the Jabber extensions of* disco *(discovery) and* privacy lists*.

13.2.5. C2S

The *C2S* (Client to Server) component handles communication with Jabber clients:

- Connects to Jabber clients
- Passes packets to the SM
- Authenticates clients
- Registers users
- Triggers activity with the SM

The *C2S* component connects to the *Authentication Data Package* (authreg) in order to register and authenticate users.

13.3. Jabberd 2 Module Decomposition

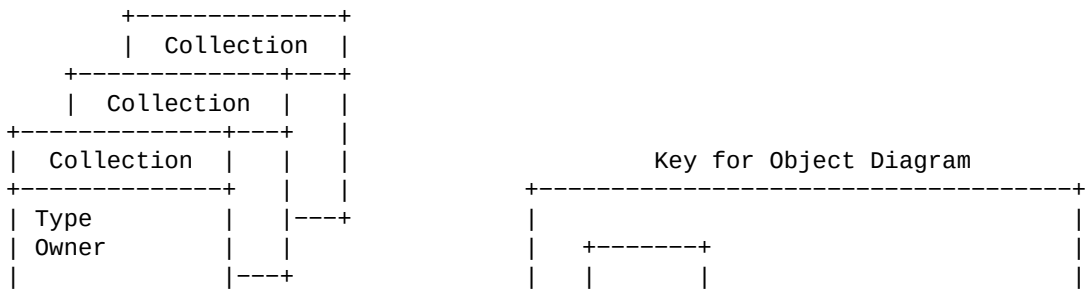
|| TODO: Section TBD ||

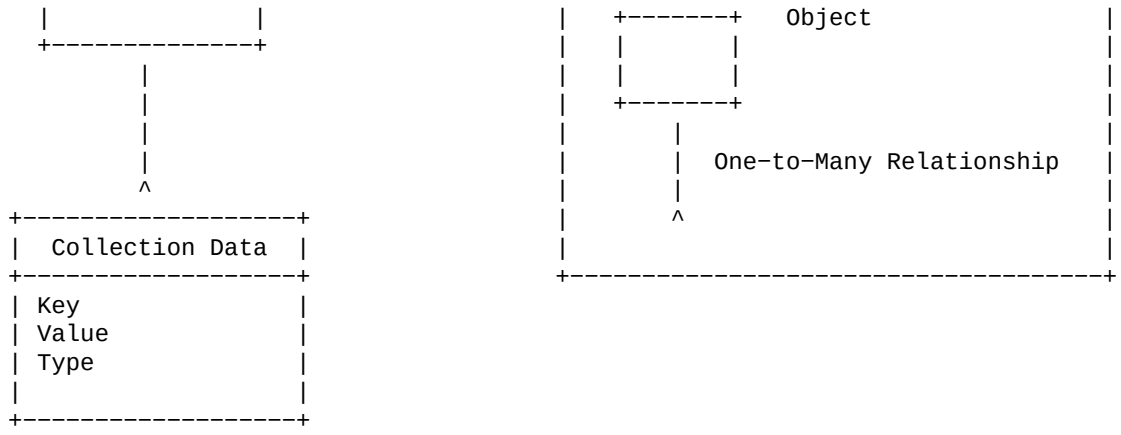
13.4. Jabberd 2 Data Handling

Jabberd employs data handling that allows for mapping to various data packages. The concept of a *collection* object lies at the core of Jabberd data handling. Each *collection* object has the attributes of *type* and *owner*. *Type* specifies what category of data is being handled, such as *queue*, *vcard* or *roster−items*. *Owner* specifies who owns the *collection*. For user−related data, *Owner* is a jabber ID (JID).

Each *collection* holds one or more data objects. Each of these data objects is a tuple consisting of a *key*, *value* and *type*. *Key* is a string that specifies the kind of data being held. *Value* is the data being held, and *type* is the is the data type, i.e. Boolean, integer, string, etc. The diagram below shows how these objects are related.

Figure 13.4. Jabberd Data Objects:





13.5. Jabberd 2 Data Structure (for MySQL)

Jabberd Data handling becomes clearer as we see it applied to the MySQL data package. In the relational world, each *collection type* becomes a table, and each *collection data key* becomes a table field. Thus, each row consists of a *collection owner* (JID) plus one or more data fields.

13.5.1 Table List (for MySQL).

The following is a list of MySQL tables for a deployment that uses Jabberd for both its authentication and its application data package (*authreg* and *db*, respectively):

active
: Stores date/time upon which each account first became active.

authreg
: Contains authentication information, including username, realm and password.

disco-items
: Stores persistent discovery information so that it is available for offline retrieval.

logout
: Stores JID and timestamp for most recent user log out.

motd-message
: Stores message of the day (MOTD) in XML format.

motd-times
: Records JID's and timestamps for receipt of MOTD.

privacy-default
: Stores the name of the current list in use for a user so it can be made active on startup.

privacy-items
: Stores user privacy lists (blacklists/whitelists).

private
: Provides private XML storage for uses such as user preferences or bookmarks.

queue
: Stores queued messages in XML format.

roster-groups
: Stores user roster items only for those roster items that have an assigned group.

roster-items
: Stores user roster items, including authorization status.

vacation-settings
: Handles vacation settings, including start, end and message.

vcard

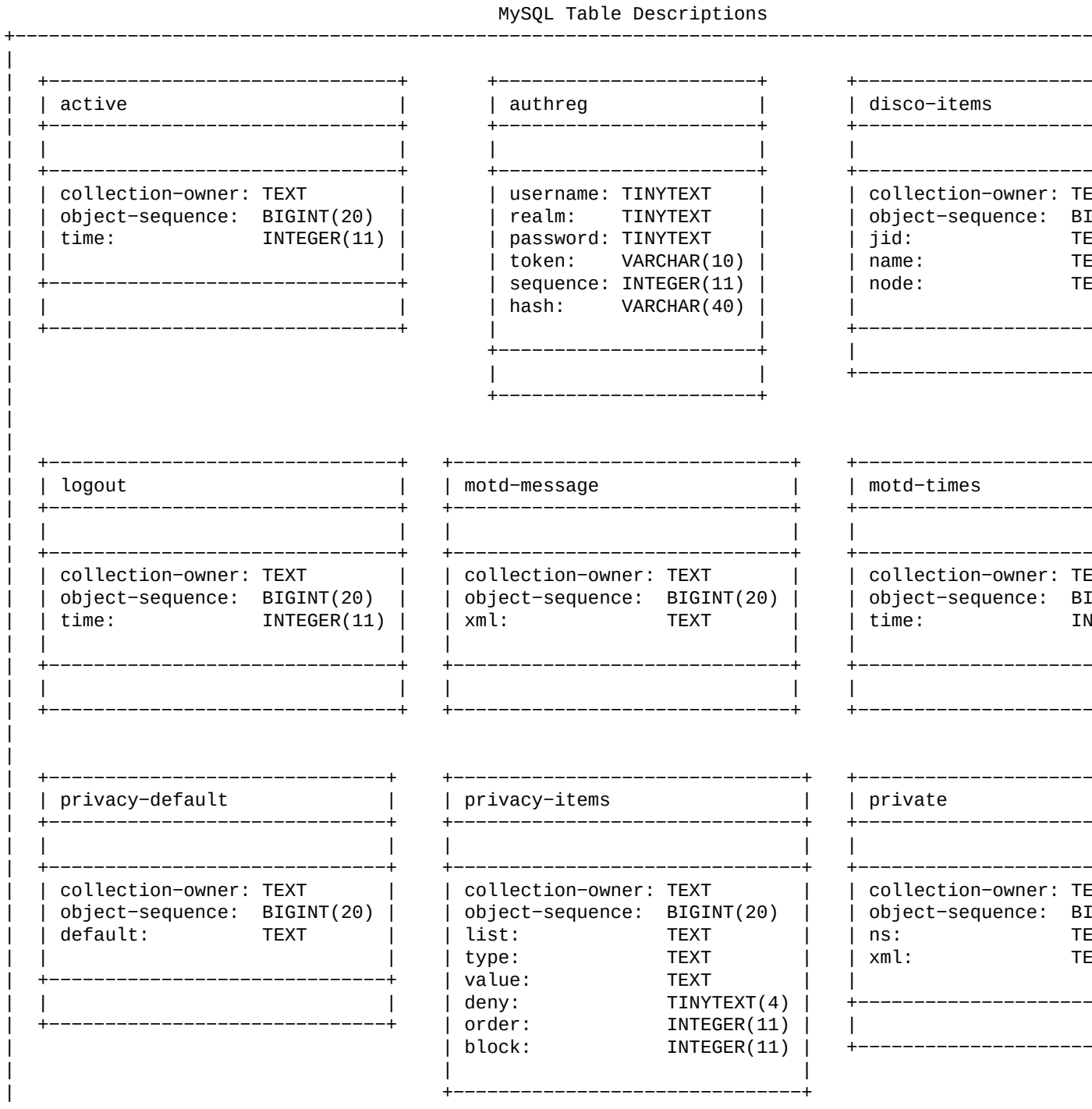
Stores vcard information.

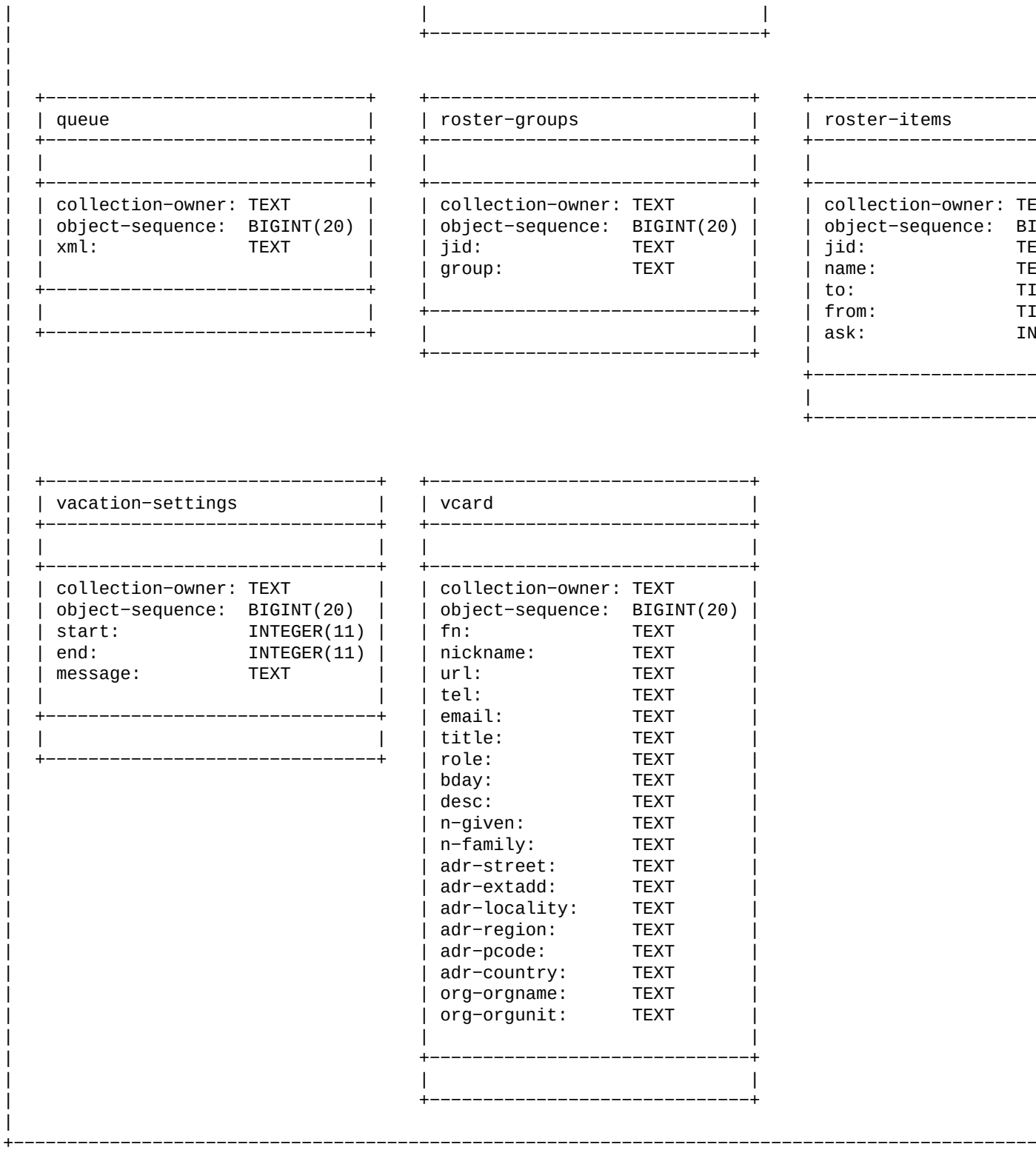
Each of the above tables represents a *collection type* in Jabberd. Note that the *authreg* table handles the authentication aspect for Jabberd when MySQL is used as the authentication data package.

13.5.2. Table Descriptions (for MySQL).

Descriptions for the Jabberd tables (as they exist in MySQL) appear in the diagram below:

Figure 13.5.2. Jabberd Table Descriptions:




queue
collection-owner: TEXT
object-sequence: BIGINT(20)
xml: TEXT
roster-groups
collection-owner: TEXT
object-sequence: BIGINT(20)
jid: TEXT
group: TEXT
roster-items
collection-owner: TE
object-sequence: BI
jid: TE
name: TE
to: TI
from: TI
ask: IN
vacation-settings
collection-owner: TEXT
object-sequence: BIGINT(20)
start: INTEGER(11)
end: INTEGER(11)
message: TEXT
vcard
collection-owner: TEXT
object-sequence: BIGINT(20)
fn: TEXT
nickname: TEXT
url: TEXT
tel: TEXT
email: TEXT
title: TEXT
role: TEXT
bday: TEXT
desc: TEXT
n-given: TEXT
n-family: TEXT
adr-street: TEXT
adr-extadd: TEXT
adr-locality: TEXT
adr-region: TEXT
adr-pcode: TEXT
adr-country: TEXT
org-orgname: TEXT
org-orgunit: TEXT

Back | Up

A.1. Jabberd 2 Quick Start Guide

This guide is intended to get Jabberd 2 installed and running as quickly as possible:

1. Install OpenSSL
2. Install Berkeley DB
3. Install Libidn
4. Create Jabber User and Group
5. Create Directories for Data and Logging
6. Install Jabberd 2
7. Configure Server
8. Test Server

Jabberd 2 can use MySQL, PostgreSQL or Berkeley DB to store its data. MySQL is the recommended database for Jabberd 2; however, this guide suggests using Berkeley DB because it requires the least amount of configuration for Jabberd 2 data storage. See the Jabberd 2 Installation and Administrative Guide for a detailed guide to installing and configuring Jabberd 2.

A.1.1. Install OpenSSL

Jabberd 2 requires that OpenSSL (version 0.9.6b or higher) be installed prior to installing Jabberd 2. If OpenSSL version 0.9.6b is not installed on your system, see the OpenSSL site or Installing OpenSSL for Jabberd.

A.1.2. Install Berkeley DB

Jabberd 2 requires Berkeley DB version 4.1.24 or higher; if this is not installed on your system, see the Berkeley DB site or Installing Berkeley DB for Jabberd 2.

Consideration should be exercised when choosing a data store for a Jabberd 2 production server because converting from one database to another may be difficult.

A.1.3. Install Libidn

Jabberd 2 requires Libidn version 0.3.0 or higher; if this is not installed on your system, see the GNU Libidn site or Installing Libidn for Jabberd 2.

A.1.4. Create Jabber User and Group

You should create a specific `jabber` user and group to run the server:

```
su
groupadd jabber
useradd –g jabber jabber
```

Note that the above commands are intended as an example. The commands and parameters for adding a user and group may vary for your system. Consult your manuals if you have any doubt about these commands.

A.1.5. Create Directories Data, Logging and PID's

You should create directories for data, logging and PID's, and your jabber user will need read and write permissions on these directories.

A.1.5.1. Create Data Directory

Create a directory to store the Jabberd database files (as superuser):

```
mkdir -p /usr/local/var/jabberd/db
```

A.1.5.2. Create Log Directory

Create a directory to store Jabberd log files (as superuser):

```
mkdir -p /usr/local/var/jabberd/log
```

A.1.5.3 Create PID Directory

Create a directory to store Jabberd PID files (as superuser):

```
mkdir -p /usr/local/var/jabberd/pid
```

A.1.5.4. Set Ownership for Data and Log Directories

Change the ownership of the directories created above (as superuser). If you used the locations specified above, enter the command:

```
chown -R jabber:jabber /usr/local/var/jabberd
```

A.1.6. Install Jabberd 2

This section describes how to download, configure, build and install Jabberd 2 on your system.

A.1.6.1. Download Jabberd

Download the file `jabberd-2.n.tar.gz` from the Jabberd 2 Releases page, where "n" is the latest version of Jabberd 2.

Download the file referenced above into a convenient directory for building the installation files. At the time of writing, Jabberd 2 stable 3 is the latest version and is used in the examples below.

A.1.6.2. Extract Jabberd Installation Files

Change to the directory where you downloaded the file above and then extract the Jabberd 2 files by running the command:

```
tar -zxvf jabberd-2.0s3.tar.gz
```

A.1.6.3 Configure the Jabberd Build

Change to the directory created above:

```
cd jabberd-2.0s3
./configure --enable-db  --disable-mysql
```

The command above will configure your build to use Berkeley DB as the data store. You may also wish to enable debugging. To do this, add `--enable-debug` to the configuration options above. For help with configuration options, enter `./configure --help`.

A.1.6.4. Build Jabberd

Build Jabberd buy running the command:

```
make
```

A.1.6.5. Install Jabberd

Switch to the super-user:

```
su
```

Run make install:

```
make install
```

A.1.5.6. Default File Locations

Your Jabberd 2 installation is complete. Below is a listing of file locations for the default installation:

```
/usr/local/etc    Jabberd Configuration Files
/usr/local/bin    Jabberd Binaries (jabberd, c2s, resolver, router, s2s, sm)
```

A.1.7. Configure Server

The most basic Jabberd configuration (when using Berkeley DB) requires a total of 4 configuration edits:

1. Set *hostname ID* in `c2s.xml`
2. Set *authreg module* to use in `c2s.xml`
3. Set *hostname ID* in `sm.xml`
4. Set *storage module* to use in `sm.xml`

The configuration files are all found in `/usr/local/etc/jabberd`.

A.1.7.1. Set hostname ID in `c2s.xml`

In `c2s.xml` edit the `id` tag under the section labelled `local network configuration` so that the `id` references a resolvable network name for your jabber server. For example, using the FQDN of `somemachine.somedomain.com`, your `c2s.xml` configuration would appear as below:

```
<local>
  <!-- Who we identify ourselves as. This should correspond to the
       ID (host) that the session manager thinks it is. You can
       specify more than one to support virtual hosts, as long as you
       have additional session manager instances on the network to
       handle those hosts. The realm attribute specifies the auth/reg
       or SASL authentication realm for the host. If the attribute is
```

```
        not specified, the realm will be selected by the SASL
        mechanism, or will be the same as the ID itself. Be aware that
        users are assigned to a realm, not a host, so two hosts in the
        same realm will have the same users.
        If no realm is specified, it will be set to be the same as the
        ID. -->
    <id>somemachine.somedomain.com</id>
```

Note that this `id` must be resolvable by the clients that will be connecting to your Jabberd server. Using an IP address as the `id` should work; however, this is strongly discouraged.

A.1.7.2. Set authreg module to use in `c2s.xml`

Further down in `c2s.xml` is a section labeled `Authentication/registration database configuration`. This is where the *authreg* data module is specified. Edit the `module` tag so that Berkeley DB is specified as the `Backend module to use`. Throughout Jabberd configuration, Berkely DB is abbreviated to `db`. Therefore, you should edit the `module` tag as below:

```
<!-- Authentication/registration database configuration -->
<authreg>
  <!-- Backend module to use -->
  <module>db</module>
```

A.1.7.3. Set hostname ID in `sm.xml`

At the top of `sm.xml` is the `id` setting for hostname. Edit the `id` tag with the same hostname specified in section A.1.6.1 above:

```
<sm>
  <!-- Our ID on the network. Users will have this as the domain part of
       their JID. If you want your server to be accessible from other
       Jabber servers, this ID must be resolvable by DNS.s
       (default: localhost) -->
  <id>somemachine.somedomain.com</id>
```

A.1.7.4. Set storage module to use in `sm.xml`

Further down in `sm.xml` is a section labeled `Storage database configuration`. This is where the *storage* data module is specified. Edit this section as below so that Berkely DB is specified as the *storage* data driver to use:

```
  <!-- Storage database configuration -->
  <storage>
    <!-- By default, we use the MySQL driver for all storage -->
    <driver>db</driver>
```

A.1.8. Test Server

If you have created the logging and data directories specified above, you should be able to start your Jabberd 2 server and connect to it.

A.1.8.1. Starting Jabberd 2

Start your Jabberd 2 server by using the start up script:

```
su
```

```
su jabber
/usr/local/bin/jabberd
```

If your server fails to start, you can start Jabberd 2 with the debug option (note that this requires building Jabberd 2 with the debug option see section A.1.5.3 above):

```
/usr/local/bin/jabberd -D
```

A.1.8.2. Connecting to your Jabberd Server

You should now be able to connect to your Jabberd server, and create a new account. Your account name should be something like `someuser@somemachine.somedomain.com`. If you are unable to connect, make certain that the hostname used (sections A.1.6.1 and A.1.6.3) is resolvable from the machine on which the client is running.

A.1.9. Further Configuration

Once your server is running, you may wish to configure it further. See especially Section 5, Common Configuration Tasks, for further information about Jabberd configuration.

Up

A.2. Installing OpenSSL for Jabberd 2

Jabberd 2 requires that OpenSSL (version 0.9.6b or higher) be installed prior to installing Jabberd 2. This appendix describes how to download the OpenSSL source in order to build and install it for Jabberd 2.

A.2.1. Download OpenSSL Installation File

Download the file `openssl-0.9.nz.tar.gz` from the OpenSSL site, where "nz" is the latest stable version of OpenSSL. At the time of writing, OpenSSL 0.9.7b' is the most current OpenSSL version and is used in the examples below.

Note that the file above contains encryption software. Follow your local laws if you download it.

A.2.2. Extract OpenSSL Installation Files

Change to the directory where you downloaded the file and then extract the OpenSSL files by running the command:

```
tar -zxvf openssl-0.9.7b.tar.gz
```

A.2.3. Build OpenSSL

Change to the OpenSSL directory just created:

```
cd openssl-0.9.7b
```

Configure OpenSSL for your system:

```
./config --prefix=/usr
```

By default, OpenSSL installs to `/usr/local/openssl`. Using the `--prefix=/usr` option causes the OpenSSL libraries to be installed to the `/usr` directory. Using this option makes OpenSSL more accessible not only to Jabber, but also to other applications. The `shared` option causes the build to create shared libraries. Read the `INSTALL` file if you are in doubt about options for your system.

Build OpenSSL on your system:

```
make
```

You can test your OpenSSL build if you wish:

```
make test
```

There should be no errors in the output.

A.2.4. Install OpenSSL

Switch to the super−user:

```
su
```

Install OpenSSL on your system:

```
   make install
OpenSSL installation is now complete &#8212; unless the 'make install' output shows errors.
```

Up

A.3. Installing Berkeley DB for Jabberd 2

Berkeley DB (version 4.1.24 or higher) is one of the three database systems that can be configured to provide data storage for Jabberd 2. This appendix describes how to download the Berkeley DB source in order to build and install if for Jabberd 2. The advantage of using Berkeley DB for Jabberd data storage is that Berkeley DB requires no configuration, maintenance or administration after it is installed.

A.3.1. Download Berkeley DB Installation Files

Download the file `db−4.1.nn.tar.gz` from the Berkeley DB Downloads page, where "nn" is the latest version of Berkeley DB. At the time of writing, Berkeley DB 4.1.25 is the most current version, and is used in the examples below.

Note that if you choose a version with strong encryption, you should follow your local laws if you download it.

A.3.2. Extract Berkeley DB Files

Change to the directory where you downloaded the file above and then extract the Berkeley DB files by running the command:

```
tar −zxvf db−4.1.25.tar.gz
```

At the time of writing, there is a patch for Berkeley DB version 4.1.25. You can apply this patch by first downloading the patch to the directory where you extracted the Berkeley DB files, for example `db-4.1.25`. Then run `patch -p0 < patchfile` to apply the patch, for example:

```
patch -p0 < patch.4.1.25.1
```

A.3.3. Build Berkeley DB

Change to the `build_unix` directory of the Berkeley DB directory just created:

```
cd db-4.1.25/build_unix
```

Configure Berkeley DB for your system. Berkeley DB uses its directory structure to identify the operating system type, so the "configure" command is different from most:

```
../dist/configure --prefix=/usr
```

The `prefix` command will install the package under `/usr`, where it will be easier to find by other applications.

Note that the default configuration should make Berkeley DB work with Jabberd 2 enabling Berkeley DB optimizations may create compatibility issues for Jabberd 2.

Build Berkeley DB on your system:

```
make
```

A.3.4. Install Berkeley DB

Switch to the super-user:

```
su
```

Install Berkeley DB on your system:

```
make install
```

Berkeley DB installation is now complete unless the `make install` output shows errors. If you receive errors, refer to documentation on the Berkeley DB site.

Up

A.4. Installing MySQL for Jabberd 2

This appendix describes how to install MySQL for Jabberd 2. MySQL is one of the three database systems that can be configured to provide data storage for Jabberd 2. The advantage of using MySQL is that it is fast and robust. Additionally, there are good documentation and support for MySQL.

A.4.1. Install from Source or Binaries

Pre−compiled RPM's are available, and the MySQL maintainers recommend that RPM's be used for installing MySQL. See MySQL 4.0 Downloads for the RPM's. Read the note below, if you install from RPM's. Otherwise, skip to A.4.2. to begin installing from the source code.

***I** Important: Minimum MySQL Version*
Jabberd 2 requires MySQL version 4.0 or higher.

***I** Important: Required MySQL RPM's*
Jabberd 2 requires that the following 4 RPM's be installed: Server; Client Programs; Libraries and Header files; and Dynamic Client Libraries.

A.4.2. Create MySQL User and Group

You will want to create a `mysql` user and group to run the MySQL server:

```
su
groupadd mysql
useradd −g mysql mysql
```

***I** Important: Check Your User and Group Commands*
The above commands are intended as an example. The commands and parameters for adding a user and group may vary for your system. Consult your manuals if you have any doubt about these commands.

A.4.3. Download MySQL Installation Files

Download the file `mysql−4.0.15a.tar.gz` from the MySQL Downloads page, where "nn.nn" is the latest version of MySQL. This download file is listed as the source tarball downloads near the bottom of the page. At the time of writing, MySQL 4.0.15 is the most current version, and is used in the examples below.

A.4.4. Extract MySQL Files

Change to the directory where you downloaded the file above and then extract the MySQL files by running the command:

```
tar −zxvf mysql−4.0.15.tar.gz
```

A.4.5. Build MySQL

Change to the `build_unix` directory of the MySQL directory just created:

```
cd mysql−4.0.15
```

Configure MySQL for your system:

```
./configure −−prefix=/usr
```

The `prefix` command will install MySQL under `/usr`, where it will be more easily found by other applications.

Build MySQL on your system:

```
make
```

A.4.6. Install MySQL

Switch to the super−user:

```
su
```

Install MySQL on your system:

```
make install
```

If you receive errors, refer to the MySQL Documentation site.

A.4.7. Set Root Password for MySQL

MySQL uses its own passwords, which are completely separate from the operating system passwords. When MySQL is installed, the root MySQL password is blank, so you should create a root MySQL password as soon as you install MySQL. Use the following command (with your own password) to create the password (as superuser):

```
/usr/local/bin/mysqladmin −u root password 'new−password'
```

A.4.8. Copy Preference File to /etc

The MySQL install process does not automatically install a MySQL preference file to `/etc/`. Instead, sample configuration files are contained in the `support−files` directory of the source code. Choose one that fits your requirements (large, medium or small installation), and copy it to `/etc` so that the preference file has the name `my.cnf` (as superuser):

```
cd support files
cp my−small.cnf /etc/my.cnf
```

A.4.9. Set Ownership for Data Directories

Set the ownership for the MySQL data directories using the user and group created above (as superuser):

```
chown −R mysql:mysql /usr/local/var/mysql/
```

A.4.10. Create Symlinks for Shared Libraries

Create symlinks for the MySQL shared libraries so that Jabberd 2 can locate them (as superuser):

```
ln −s /usr/local/include/mysql/ /usr/include/mysql
ln −s /usr/local/lib/mysql/ /usr/lib/mysql
```

A.4.11. Set MySQL to Start at Boot

Set the the MySQL daemon to start at boot as the mysql user created above. Boot scripts vary by distribution and architecture and so are beyond the scope of this guide. Consult your user documentation to set `/usr/local/libexec/mysqld` to start at boot time.

A.4.12. Test MySQL

Your MySQL server is ready to be tested. Start the server (as superuser):

```
/usr/local/bin/mysqld_safe --user=mysql &
```

You should now be able to connect to the server as the MySQL root users:

```
/usr/local/bin/mysql
```

This should provide a `mysql>` prompt. You can test your installation with a show databases command:

```
mysql> SHOW DATABASES;
```

You should see output like this:

```
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)
```

MySQL is now successfully installed on your system.

Up

A.5. Installing Libidn for Jabberd 2

Libidn provides necessary string manipulation functionality for Jabberd 2. Prior to Jabberd 2 stable 3, libidn was included with the Jabberd 2 distribution; however, a licensing conflict makes it necessary that libidn be installed separately.

A.5.1. Download Libidn Installation Files

Download the file `libidn-0.n.n.tar.gz` from the GNU Libidn Page page, where "n.n" is the latest version of Libidn. At the time of writing, Libidn libidn-0.5.0 is the most current version, and is used in the examples below.

A.5.2. Extract Libidn Files

Change to the directory where you downloaded the file above and then extract the Libidn files by running the command:

```
tar -zxvf libidn-0.5.0.tar.gz
```

A.5.3. Build Libidn

Change to the `libidn-0.5.0` directory just created:

```
cd libidn-0.5.0
```

Configure Libidn for your system. The `--prefix=/usr` option will cause the libraries to be installed under `/usr/lib`, making them easier for other applications to find:

```
./configure --prefix=/usr
```

Build Libidn on your system:

```
make
```

A.5.4. Install Libidn

Switch to the super-user:

```
su
```

Install Libidn on your system:

```
make install
```

Libidn installation is now complete unless the `make install` output shows errors. If you receive errors, refer to documentation on the Libidn site.

Up

A.6. Generating a Self-Signed SSL Certificate

This appendix describes how to generate a self-signed OpenSSL certificate for use with Jabberd.

I *Important: Key Is Self-Signed*

The key generated by the instructions below is *self-signed*. Such a key is not part of a trust hierarchy. When used to secure communications with Jabber clients, a self-signed key will usually cause warnings to appear because its authenticity cannot be verified against a trusted key.

A.6.1. Generate Key Pair

From a working directory, enter the command below to begin an interactive key generation process:

```
openssl req -new -x509 -newkey rsa:1024 -days 3650 -keyout privkey.pem -out server.pem
```

You will be prompted for a passphrase for the private key. After entering and confirming your passphrase, you will be prompted for public information about your key.

N *Note: Common Name*

Note that you should enter your domain name as the `Common Name` for your certificate.

N *Note: Key Lifetime*

Note that the command above creates a key with a 3650 day (10 year lifetime). To change the key lifetime, use a different number of days for the `-days` parameter.

A.6.2. Remove Passphrase

Enter this command to remove the passphrase from your private key:

```
openssl rsa −in privkey.pem −out privkey.pem
```

A.6.3. Combine the Private and Public Key

Enter this command to combine the private and public keys into a single file:

```
cat privkey.pem >> server.pem
```

A.6.4. Delete Private Key

You should now delete your private key:

```
rm privkey.pem
```

A.6.5. Move Key and Set Permissions

You can now move your key to its permanent location. For example, to move the key to the default Jabberd `pemfile` location, you would enter this command (as superuser):

```
mv server.pem /usr/local/etc/jabberd/server.pem
```

Then, you should set permissions on this file so that it is owned by superuser and is readonly (as superuser):

```
chown root:jabber /usr/local/etc/jabberd/server.pem
chmod 640 /usr/local/etc/jabberd/server.pem
```

Your certificate is now ready for use by Jabberd. You should make a backup (such as to a floppy) of your certificate.

Up

A.7. Jabberd for Corporate Use

This appendix provides some tips for installing Jabberd for use by a corporation or other private organization. These tips were originally posted by Ken Wermann on the Jadmin Mailing List :

- Install from Source
- Install to Unix
- Locate Server in DMZ or on LAN
- Enable Only SSL Communication
- Apply OpenSSL Patches
- Disable User Registration
- Use Strong Server Passwords
- Avoid Transports and Additional Services
- Log Conversations
- Use JAJC or Exodus Jabber Client

These tips are not meant as an exhaustive guide to installing Jabberd for corporate use.

A.7.1. Install from Source

Install the latest stable Jabberd server from source. Although there are many packages (RPM's, etc) available for Jabberd, users report various problems with some of these. At the time of writing, Jabberd 2.0s1 is the latest stable release, and it is available from Jabber Studio.

A.7.2. Install to Unix

Jabberd is native to Unix. Although libraries exist for installing Jabberd to Windows, you should make your corporate installation only to a flavor of Unix.

A.7.3. Locate Server in DMZ or on LAN

Administrators should give careful consideration to whether users may need to connect from outside the corporate firewall. If users will *never* need to connect from outside the firewall, then locate the Jabberd server on the corporate LAN. For the Jabberd host name, you can use any name that clients can resolve. Your Jabberd server will not be able to communicate with other Jabber servers.

If you have users that need to connect from outside the firewall even if only occasionally you should locate your Jabberd server in a DMZ. Open port 5223 on both the DMZ and firewall to permit SSL encrypted Jabber communication. For your Jabberd host name, you can use a host on your domain, such as `jabber.mycompany.com`.

A.7.4. Enable Only SSL Communication

Even if your server is located on the corporate LAN, you should enable only SSL communications. Instant messaging traffic is easy to sniff on a network. See Section 5.2 for information about how to configure Jabberd 2 for SSL.

A.7.5. Apply OpenSSL Patches

Keep your OpenSSL installation up to date and apply the latest patches as they become available. You may wish to subscribe to the OpenSSL Mailing Lists.

A.7.6. Disable User Registration

Public user registration should be disabled. See Section 5.5 for information about how to disable public registration for Jabberd 2. Enabling user password change (Section 5.6) would be a good idea.

With public registration disabled, an administrator(s) will need to create user accounts. See the JabberStudio: Script Repository for user creation scripts that can be used with Jabberd 1.4. See Section 6.3 for information about how to create accounts with Jabberd 2.

A.7.7. Use Strong Server Passwords

Make certain that you change all default server passwords and secrets, and you should use strong passwords for these. These passwords include the password for your database connection and the password for your router connections.

A.7.8. Avoid Transports

Although Jabber transports (for foreign IM systems) can provide desirable features, administrators should avoid providing these services because they may create additional security vulnerabilities in addition to HR risks. On the other hand, JUD and Conferencing are recommended services for corporate installations. Note that at the time of writing JUD does not function properly with Jabberd 2.

A.7.9. Log Conversations

Check with you legal department to determine the requirements for message logging. Laws about message logging vary by country, state, province, etc., and these laws are often prescribe how long message logs must be preserved. This is especially true for certain industries, such as financial and healthcare.

Log IM messages if this is permitted or required. Inform your users that their conversations are being logged because this is good practice and because this will discourage inappropriate use of IM. See JabberStudio for utilities for monitoring Jabber traffic. Currently, there are no available utilities for logging messages on Jabberd 2.

A.7.10. Use JAJC or Exodus Jabber Client

JAJC is a good choice of a client for deployments running Jabberd 1.4 on Windows 2000 or higher because it is a mature, full−featured Jabber client that supports SSL.

Exodus is a good choice of a client for deployments running Jabberd 2. Like JAJC, Exodus is a stable client with many Jabber features. Additionally, Exodus supports SASL for authentication and TLS for channel encryption. Exodus is also a good choice for a clients running on older Windows systems.

Up

A.8. Automatic Startup and Shutdown Using an RC Script

SysV initialization scripts and process controls provide one method of automatically starting and stopping Jabberd 2 when the machine comes up and goes down. This appendix describes how to implement SysV control for Jabberd on the Fedora 1.0 platform. To modify this process for your own distro, please consult your Linux or Unix documentation.

A.8.1. Download RC Script

An RC script for Jabberd 2 currently exists in the tools directory of the Jabberd 2 CVS Repository. Download the file `jabberd.rc` from CVS and copy it to the `/etc/rc.d/init.d` directory.

A.8.2. Rename RC Script

Rename the downloaded file to 'jabberd2':

```
mv −f /etc/rc.d/init.d/jabberd.rc /etc/rc.d/init.d/jabberd2
```

A.8.3. Assign Ownership and Permissions

The RC script file should be executable and owned by root:

```
chown −f root:root /etc/rc.d/init.d/jabberd2
chmod −f 0755 /etc/rc.d/init.d/jabberd2
```

A.8.4. Create Symbolic Links

You should create start and kill symbolic links in all appropriate run level directories:

```
ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc0.d/K15jabberd2

ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc1.d/K15jabberd2

ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc2.d/S85jabberd2
ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc2.d/K15jabberd2

ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc3.d/S85jabberd2
ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc3.d/K15jabberd2

ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc4.d/S85jabberd2
ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc4.d/K15jabberd2

ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc5.d/S85jabberd2
ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc5.d/K15jabberd2

ln −s /etc/rc.d/init.d/jabberd2 /etc/rc.d/rc6.d/K15jabberd2
```

A.8.5. Restart Jabberd 2

You should now be able to test your RC script:

```
/etc/rc.d/init.d/jabberd2 restart
/etc/rc.d/init.d/jabberd2 restart
```

The second invocation of restart should all result in [OK]. Note that if you execute redhat−config−services from the Fedora gui or execute ntsysv from a Fedora terminal or console session, you will now find jabberd2 in the list of services.

I *Important: Script Runs as Root*

The script, as included in the Jabberd 2 Distribution, runs as Jabberd 2 as root. A workaround for this issue is to modify the RC script so that the starting user (usually root) is `su` to the `jabber` user. Change line 125 of the script to read:

```
su − jabber −c "${progsPath}/${prog} ${args} & 2> /dev/null"
```

Up

A.9. Automatic Startup and Shutdown Using Daemontools

D.J. Bernstein's Daemontools provide another method of controlling Jabberd 2 through machine startup and shutdown. This appendix assumes that Daemontools are installed on your system. See How to Install Daemontools for installation instructions. See the Daemontools homepage for a description of Daemontools.

Daemontools configuration requires a run script in a dedicated directory. This directory is then symbolically linked to the `/service` directory for monitoring by the Svscan daemon of Daemontools. This how−to is a bit long because each of the Jabberd 2 binaries must have its own linked directory and run script. Additionally, the PID file writing should be disabled (in Jabberd XML configuration) for each of the binaries.

A.9.1. Create Master Directory

Create a master directory for the 5 Daemontools directories to be created below. The most convenient place for this directory is in `/usr/local/etc/jabberd/`. Create this new directory (as root):

```
mkdir /usr/local/etc/jabberd/daemontools
```

A.9.2. Create Run Directories for Binaries

Create a sub−directory for each of the Jabberd binaries:

```
mkdir /usr/local/etc/jabberd/daemontools/router

mkdir /usr/local/etc/jabberd/daemontools/resolver

mkdir /usr/local/etc/jabberd/daemontools/sm

mkdir /usr/local/etc/jabberd/daemontools/c2s

mkdir /usr/local/etc/jabberd/daemontools/s2s
```

You should now have 5 sub−directories under `/usr/local/etc/jabberd/daemontools/`.

A.9.3. Create Run Scripts

Each of the sub−directories now needs a run script for its respective Jabberd binary. Change to the `router` sub−directory to create your first run script:

```
cd /usr/local/etc/jabberd/daemontools/router
```

Using an editor, create a file called `run` in this directory. Edit the file as below so that it contains a run script for the router component:

```
#!/bin/sh
exec setuidgid jabber /usr/local/bin/router /usr/local/etc/jabberd/router.xml
```

Repeat this process for each of the remaining 4 sub−directories. You can use the above text as a template, and replace the 2 instances of `router` with the respective Jabberd binary (resolver, sm, c2s and s2s). For example, the `run` file in the `resolver` sub−directory should be created as below:

```
#!/bin/sh
exec setuidgid jabber /usr/local/bin/resolver /usr/local/etc/jabberd/resolver.xml
```

A.9.4. Make Run Scripts Executable

Each of the run scripts above needs to be executable. Run the command below to make each of the run scripts executable:

```
chmod 0755 /usr/local/etc/jabberd/daemontools/*/run
```

A.9.5. Disable PID's for each of the Jabberd Binaries

Svscan does not rely on PID files; therefore, PID file writing should be disabled in each of these configuration files:

```
router.xml
resolver.xml
sm.xml
c2s.xml
s2s.xml
```

Near the top of each of these files is a tag set for the PID file location. Comment these tags in each file above to disable PID file writing. For example, the top of `router.xml` should appear as below:

```
<!-- Router configuration -->
<router>
  <!-- ID of the router on the network (default: router) -->
  <id>router</id>

  <!-- The process ID file. comment this out if you don't need to know
       to know the process ID from outside the process (eg for control
       scripts) -->
  <!-- <pidfile>/usr/local/var/jabberd/pid/router.pid</pidfile> -->
```

A.9.6. Create Symbolic Links

Each of the sub−directories now needs to be linked to the `/service/` directory. Once these symlinks are created, svscan will start and monitor the respective process.

I *Important: Stop Jabberd Server Before Proceeding*
In order to prevent conflicts, you should make certain that your Jabberd server is completely stopped before proceeding with the final step of Daemontools configuration.

Create 5 symlinks as below:

```
ln −s /usr/local/etc/jabberd/daemontools/router /service

ln −s /usr/local/etc/jabberd/daemontools/resolver /service

ln −s /usr/local/etc/jabberd/daemontools/sm /service

ln −s /usr/local/etc/jabberd/daemontools/c2s /service

ln −s /usr/local/etc/jabberd/daemontools/s2s /service
```

A.9.7. Check that Services are Running

Make sure that the `svscan` daemon is running on your machine. On my distro (Gentoo Linux), the `svscan` daemon is started from `/etc/init.d`. You should now be able to check the status of your Jabberd services by using the `svstat` command. For example, to check the status of the router component, you would switch to the `/service` directory and then enter the `svstat` command as below:

```
svstat router
```

You should see output similar to this:

```
router: up (pid 20011) 248 seconds
```

Svscan will now monitor your Jabberd processes, and if a process goes down, Svscan will restart it.

***N** Note: Svscan Should Be Set to Start at Boot*

If you just installed Daemontools, you may need to add the svscan daemon to an appropriate run level so that it starts during boot. Consult your distro's documentation for the best way to do this.

Up

A.10. Using Jabber for Linux System Monitoring

This section describes how Jabber can easily be extended for Linux system monitoring. Information in this section is not particular to Jabberd 2; however, I have included it in the guide because I have created some system administration tools that use Jabber, and I hope that others may find these scripts useful. Additionally, my scripts may possibly inspire readers to extend Jabber in new ways.

Jabber is ideally suited for system administration alerts because Jabber provides real time notification. Jabber alerts are less likely to be ignored than email alerts, they can be received anywhere the user is logged in, and Jabber alerts are free, unlike pagers.

Each subsection of this appendix describes a way that Jabber can be used for Linux system administration monitoring and alerting:

- Using the jabber_alert.pl Script
- Using Jabber for Job Monitoring
- Using Jabber for System Monitoring
- Using Jabber for System Debugging
- Using Jabber to Receive Time Sensitive Email
- Using Jabber with Mon and Nagios

All of the above tasks rely on scripts available on this site. Impatient readers may wish to peruse scripts in the tools folder before continuing.

A.10.1. Using the jabber_alert.pl Script

The jabber_alert.pl script is a Perl script that sends a pre−formatted alert message via Jabber. It accepts a variety of arguments that are parsed as alert headers for a Jabber message; however, in its simplest usage it can be used to send a Jabber message from the command line:

```
./jabber_alert.pl −e user1@somedomain.com −n user2@somedomain.com −w user2passwd

Where −e [recipient JID] −n [sender JID] −w [sender password] [message body from STDIN]
```

The message body comes from standard input, and can contain as many lines as you wish. When you are ready to send the message, press `CTRL−d`. The `−h` option displays help information.

The jabber_alert.pl script can be downloaded from the tools folder. Once downloaded, you will need to chmod it (`chmod a+x jabber_alert.pl`) so that you can run it. `jabber_alert.pl` relies on these three Perl packages:

```
Net::Jabber
Time::Local
Getopt::Std
```

Your Perl installation probably has `Time::Local` and `Getopt::Std` installed, and the script will halt if you do not. You can download the Net::Jabber install file from the CPAN.org web site. To install it, first untar it and then build it:

```
perl Makefile.PL
make
make install
```

N *Note: CPAN Auto−Installer*

You may wish to setup and use Perl's CPAN.pm auto−installer to install Perl modules. `CPAN.pm` automates the download, make and install of Perl modules and extensions. To setup `CPAN.pm`, enter the command `perl −MCPAN −e shell` (as super user). This starts an interactive script that will set up your machine to fetch and install Perl modules from the command line. Once setup, you would enter the command `install Net::Jabber` (from the `cpan` shell) to install `Net::Jabber`. One of the benefits of using `CPAN.pm` is that it will also fetch dependencies.

Once `Net::Jabber` is installed, you can use `jabber_alert.pl` to send Jabber messages from the command line; however, its real purpose here is to work as a Jabber alert message sender for various bash shell scripts. That is, once `jabber_alert.pl` is installed, it is easy to send a Jabber alert message from a shell script. *All* of the individual shell scripts in this section rely on `jabber_alert.pl`.

N *Note*

To use it from the command line, the user password must be entered in the clear; therefore, you should be careful when using it this way, or you should use a jabber account that is not valuable.

Once `jabber_alert.pl` is installed, you can check out the system administration scripts I have written that use it, starting with the next section.

Up	Next

A.10.2. Using the job_mon.sh Script to Monitor Jobs

This section describes how to use the job_mon.sh script to monitor jobs, such as cron jobs. `job_mon.sh` is a script that runs a job. If the job fails (exits with a status other than 0), `job_mon.sh` sends an alert message via Jabber. The alert message contains information about the job and any information that the job sent to STDERR. Optionally, `job_mon.sh` can provide an alert message if the job completed successfully. Its usage is simple:

```
./job_mon.sh −j [job with or without args]
```

The `−j` argument specifies the job or program for `job_mon.sh` to run. The job and any job arguments should be enclosed (as a single string) in quotation marks. An optional `−s` flag specifies that an alert should be sent on success. The `−h` option provides a help message.

A.10.2.1. Setup

The job_mon.sh script can be downloaded from the tools folder. Once downloaded, you will need to chmod it (`chmod a+x job_mon.sh`) in order to run it. `job_mon.sh` relies on jabber_alert.pl.

`job_mon.sh` requires setup for the JID of the alert sender and recipient, in addition to a password for the sender. Open `job_mon.sh` in an editor, and you will see the user configuration near the top of the file. You should edit the file to provide a `recipient_jid`, `sender_jid` and `sender_pw`. Note that `recipient_jid` may be the same as `sender_jid`. You may enter other optional information below.

***N** Note*

To use `job_mon.sh`, the script file must contain a Jabber user password. See the warning in the help message.

A.10.2.2. Testing

Once you have edited your Jabber information in `job_mon.sh`, you can easily test it. Choose a simple program to run, such as a text editor, and launch it with `job_mon.sh` :

```
./job_mon.sh −j "nedit"
```

Your program should start. Now, from another terminal, kill the program you started with 'job_mon.sh':

```
ps −A | grep nedit
>25430 pts/1    00:00:00 nedit
kill 25430
```

You should receive a Jabber message that looks something like this:

```
[13:03:38] <monitor_user@somedomain.com>
Alert: Alert Message
Time: Wed, 28 Jul 13:08:48 UTC: −4.00
User: me
Host: machine.somedomain.com
Service: nedit
Status: Job Failed: nedit terminated
with signal 15 (SIGTERM)
------------Beg−Summary------------
------------End−Summary------------
Subject: Monitor Alert
```

A.10.3. Examples

I use `job_mon.sh` to monitor my backup jobs. For example, I run a weekly backup to tape. My fcron entry uses `job_mon.sh` :

```
0 2  * * 6      /home/scripts/job_alert.sh −s −j "/home/scripts/tape_backup_full.sh"
```

The morning after the backup is run, I receive an alert message for successful run or failure. For example, this script recently produced a Jabber success alert:

```
[03:15:39] <monitor_user@somedomain.com>
Alert: Alert Message
Time: Sun, 25 Jul 03:15:38 UTC: −4.00
```

```
User: me
Host: myhost.mydomain.com
Service: /home/scripts/tape_backup_full.sh
Status: Job Completed Successfully
------------Beg-Summary------------
Rewinding tape
Tape rewind succeeded.
Mounting /boot
Boot mount succeeded.
Starting backups to tape...
Backing up: /home/
star: '/home/./me/.kde3.1/kdeinit-:0' unsupported file type 'socket'. Not dumped.
star: '/home/./me/.SB-sock' unsupported file type 'socket'. Not dumped.
star: 47590 blocks + 0 bytes (total of 1559429120 bytes = 1522880.00k).
star: The following problems occurred during archive processing:
star: Cannot: stat 0, open 0, read/write 0. Size changed 0.
star: Missing links 0, Name too long 0, File too big 0, Not dumped 2.
star: Processed all possible files, despite earlier errors.
Backups complete. Unmounting /boot
Boot unmount succeeded.
Backups complete!
------------End-Summary------------
Subject: Monitor Alert
```

`job_mon_alert.sh` catches all output that the job sends to STDERR, and this output is displayed in the summary section of the alert. If you are using `job_mon_alert.sh` with your own shell scripts, you may wish to pipe STDOUT to STDERR so that the alert message captures everything your script would output to STDOUT. For example, the `echo` command as piped below would output STDERR:

```
echo "Backups complete!" >&2
```

When a script containing this piped command is run, the text "Backups complete!" would output to the tty and it would be captured as STDERR by `job_mon.sh`. Note that job success or failure is not determined by whether the job produces any error output. Rather, `job_mon_alert.sh` reports failure or success based on the exit status of the job that is run.

| Back | Up | Next |

A.10.3. Using the sys_mon.sh Script to Monitor System Resources

This section describes how to use the sys_mon.sh script to receive real time Jabber alerts when a system resource is above a preset limit. When run, `sys_mon.sh` checks a specific system resource and checks whether that resource is above the preset threshold. Usage is based on which resource is to be checked:

```
sys_mon.sh -u|d|m [limit param(s)]
```

The `-u` argument specifies a file system usage check. The limit param consists of an integer that represents the percentage full. Thus, when run with the `-u` argument, `sys_mon.sh` will alert for all mounted file systems over '[limit percentage]' full.

The `-d` argument specifies a check on directories, where the '[limit params]' consist of threshold of kilobytes of disk space used, followed by directory(ies) under which to check, and optionally, directory(ies) to skip. Thus, when run with the `-d` argument, `sys_mon.sh` will alert for directories using more than the the threshold kilobytes.

The `-m` argument specifies a check for modified files, where the '[limit params]' consist of a number of minutes, followed by directory(ies) under which to check, and optionally, directory(ies) to skip. Thus, when run with the `-m` argument, `sys_mon.sh` will alert for files modified during the last (threshold) minutes under the stated directory(ies).

See the help (`./sys_mon.sh -h`) for more detailed usage information.

A.10.3.1. Setup

The sys_mon.sh script can be downloaded from the tools folder. Once downloaded, you will need to chmod it (`chmod a+x sys_mon.sh`) in order to run it. `sys_mon.sh` relies on jabber_alert.pl.

`sys_mon.sh` requires setup for the JID of the alert sender and recipient, in addition to a password for the sender. Open `sys_mon.sh` in an editor, and you will see the user configuration near the top of the file. You should edit the file to provide a `recipient_jid`, `sender_jid` and `sender_pw`. You may enter other optional information below.

***N** Note*

`sys_mon.sh` must contain a Jabber user password. See the warning in the help message.

A.10.3.2. Testing

Once `sys_mon.sh` has been setup, it can easily be tested. Run `sys_mon.sh` to check for file systems over 10 percent full:

```
./sys_mon.sh -u 10
```

You should receive a Jabber alert message that reports all file systems over 10 percent used, assuming, of course, that you have a file system over 10 percent used. The alert should look something like this:

```
[13:58:04] <monitor_user@somedomain.com>
Alert: Alert Message
Time: Mon, 2 Aug 13:57:59 UTC: -4.00
User: me
Host: machine.somedomain.com
Service: File System Usage Warning
------------Beg-Summary------------

File system(s) over 10%:

/ 56%
/home 79%
/home/media 41%
/mnt/temp 87%

------------End-Summary------------
Subject: Monitor Alert
```

A.10.3.3. Examples

I use `sys_mon.sh` to monitor for modified files in `/etc` on my server. To do this, I run `sys_mon.sh` as a daily cron job, and set it to check for files modified during the past 1,440 minutes (1 day):

```
0 5 * * *   /home/scripts/sys_mon.sh -m 1440 /etc -/etc/tiny/log -/etc/tiny/supervise
```

This is not so much a security tool for me, as it is a system administration tool. I share maintenance of a box with another administrator, so the alert above lets us know if the other has made configuration updates.

I also use `sys_mon.sh` as a cron job on my workstation to catch file systems before they are completely full:

```
0 4  * * *  /home/scripts/sys_mon.sh -u 90
```

In fact, I received this alert this morning:

```
[04:00:07] <monitor_user@somedomain.com>
Alert: Alert Message
Time: Mon, 2 Aug 04:00:01 UTC: -4.00
User: me
Host: machine.somedomain.com
Service: File System Usage Warning
------------Beg-Summary------------

File system(s) over 90%:

/home 94%

------------End-Summary------------
Subject: Monitor Alert
```

And before I did anything else, I did some `/home` cleaning.

Run `./sys_mon.sh -h` for more examples.

Back	Up	Next

A.10.4. Using the sys_debug.sh script for System Debugging

This section describes how to use the sys_debug.sh script as a system diagnostic and troubleshooting tool. The `sys_debug.sh` script is designed to watch a process (or processes) and send a Jabber alert for log events or alert for consumption of high system resources.

`sys_debug.sh` differs from `sys_mon.sh` (in the previous section) in that `sys_debug.sh` is designed to run continuously as a troubleshooting tool. It monitors a process or resources that a process is using. Thus, `sys_debug.sh` would be useful for watching processes that are running abnormally, or for monitoring new daemons that are being put into production as a way to ensure that the new daemons do not consume abnormal resources. Usage is based on the type of monitoring to be performed:

```
Usage: ./sys_debug.sh -a|-f|-p [test_param(s)]

Usage Options: ./sys_debug.sh [-s cycle_seconds] [-u exit_seconds]
               [-q] [-i alert_interval] -a|-f|-p test_params
```

The `-a` argument specifies monitoring of available system RAM, where the the '[test_param]' is available system RAM in kilobytes. `sys_debug.sh` will send an alert if available RAM falls below this threshold. Technically, this does not watch a specific process; however, the intention is to ensure that a process or process group is not consuming too much memory.

The `-f` argument specifies monitoring of a log file, where '[test_param(s)]' are the full path the log file and a search string. `sys_debug.sh` will send an alert if the search string is written to the log file. On alert, the last 5 lines of the log file will be reported; however, the number of lines to report can be modified with the optional `-n` argument. `tail` and `grep` can be used to watch log output; what this script does is give you a real time remote alert when a specified string is written to a log. This would be useful for watching for intermittent errors.

The `-p` argument specifies monitoring of a process via `ps` , where '[test_param(s)]' consist of a choice (`--`pid | `--`ppid | `-`C | `--`User) of process filtering followed by a `ps` header name and threshold. `sys_debug.sh` can monitor any `ps` output that returns an integer, and it will alert when `ps` reports that a process as filtered by PID, PPID, Command Name, or User rises above the stated threshold. Multiple `ps` headers and thresholds are acceptable.

See the help (`sys_debug.sh -h`) for more detailed usage information, including information about optional arguments. See the `ps` man page for more information about `ps` headers (STANDARD FORMAT SPECIFIERS).

A.10.4.1. Setup

The sys_debug.sh script can be downloaded from the tools folder. Once downloaded, you will need to chmod it (`chmod a+x sys_debug.sh`) in order to run it. `sys_debug.sh` relies on jabber_alert.pl.

`sys_debug.sh` requires setup for the JID of the alert sender and recipient, in addition to a password for the sender. Open `sys_debug.sh` in an editor, and you will see the user configuration near the top of the file. You should edit the file to provide a `recipient_jid`, `sender_jid` and `sender_pw`. You may enter other optional information below.

When using `sys_debug.sh`, you may wish to used a fixed-width font, such as courier, in your Jabber client so that output columns align properly.

***N** Note*

`sys_debug.sh` must contain a Jabber user password. See the warning in the help message.

A.10.4.2. Testing

`sys_debug.sh` can be tested against a process running on the workstation on which it is installed by testing it against low thresholds. In this example, `sys_debug.sh` is run to check if any processes running for the user `me` are consuming more than 4% memory, 3% CPU or have caused more than 10,000 page faults:

```
./sys_debug.sh -p --User me %mem 4 %cpu 3 majflt 10000
```

When run, you should receive an alert message that looks something like this:

```
Subject: Monitor Alert
[16:56:16] <monitor_user@somedomain.com>
Alert: Alert Message
Time:  Mon, 2 Aug  16:56:14   UTC: -4.00
User:  me
Host:  machine.somedomain.com
Service:  Process Alert from ./sys_debug.sh
------------Beg-Summary------------

Thresholds for majflt, %cpu, %mem reached:
```

```
( %mem over 4 )
PID      CMD                 %mem
11227    vmware-vmx          18.4
11229    vmware-vmx          18.4
11230    vmware-vmx          18.4
11231    vmware-vmx          18.4
11232    vmware-vmx          18.4
11233    vmware-vmx          18.4
11234    vmware-vmx          18.4
11236    vmware-vmx          18.4
11237    vmware-vmx          18.4

( %cpu over 3 )
PID      CMD                 %cpu
11237    vmware-vmx          19.0

( majflt over 10000 )
PID      CMD                 majflt
4422     enlightenment       3893626
11227    vmware-vmx          25091
11237    vmware-vmx          44909

------------End-Summary------------
Subject: Monitor Alert
```

Note that `ps` does not report combined memory usage by a parent PID. As far as I know, this is a limitation of Linux versions lower than 2.6.

A.10.4.3. Examples

`sys_debug.sh` can be used to monitor a new Jabberd 2 installation. After putting a Jabberd 2 server into production use, you might run `sys_debug.sh` to watch for high memory or CPU usage:

```
./sys_debug.sh -p --User jabber %mem 20 %cpu 20
```

Run `./sys_debug.sh -h` for more examples.

Back	Up	Next

A.10.5. Using the email_alert.sh Script to Receive Time Sensitive Email

This section describes how to receive Jabber notification for time sensitive email. The email_alert.sh script forwards an email via Jabber when an arriving email matches a filter.

System administrators often use email to alert for issues that need quick resolution; however, email is not well suited for real time notifications. Another use for this script is the ability to receive important email's while you are away from your email client. For example, you may not wish to check your personal email while at work. This script would allow you to receive the most important messages via Jabber at work, while ignoring the rest until you arrive home.

The `email_alert.sh` script does not poll a POP server. Rather, it relies on a running email client to receive and sort mail into directories. `email_alert.sh` periodically checks specified directories for new email, which is then checked against a filter string. If the string is matched, the email message is forwarded

via Jabber. This script is designed only to work with email messages that are stored as individual files.

`email_alert.sh` is designed to be run continuously in the background:

```
./email_alert.sh -s [seconds] &
```

The optional `-s` argument sets the check interval and defaults to a check every 30 seconds.

A.10.5.1. Setup

The email_alert.sh script can be downloaded from the tools folder. Once downloaded, you will need to chmod it (`chmod a+x email_alert.sh`) in order to run it. `email_alert.sh` relies on jabber_alert.pl.

A.10.5.1.1. Jabber Information

`email_alert.sh` requires setup for the JID of the alert sender and recipient, in addition to a password for the sender. Open `email_alert.sh` in an editor, and you will see the Jabber configuration near the top of the file. You should edit the file to provide a `recipient_jid`, `sender_jid` and `sender_pw`. You may enter other optional information immediately below.

***N** Note*

`email_alert.sh` must contain a Jabber user password. See the warning in the help message.

A.10.5.1.2. Email Filters

Below the Jabber configuration is a section for email filters. Each filter consists of a directory and a regular expression. The intention is that you use your email client, MUA, MTA, etc., to filter your email into different directories. `email_alert.sh` can then scan one (or more) of these directories and look for new messages that match a corresponding regex.

For example, if you filter important email into a directory called "sysadmin", and you want instant notification for any email that contains the word "Important" in the subject line, you might set up a filter like this in `email_alert.sh`:

```
/home/me/Mail/sysadmin
^Subject: .*Important
```

If a matching email arrived, the email would be forwarded to you via Jabber. Note that the regex is grepped against the entire body of the email. Starting a regex with `^Subject:`, `^To:` or `^From:` helps to catch information in the message headers.

A.10.5.2. Testing

To test `email_alert.sh`, you can set it up with a broad filter, such as the one below:

```
/home/me/Mail/inbox
^To:
```

The above filter would effectively jab you for any email that arrived in your inbox. Start `./email_alert.sh`, and stop it after it begins to jab email to you.

A.10.5.3. Examples

The `email_alert.sh` script contains some filter examples. See the section labeled, "BEGIN FILTER REGEX EXAMPLES."

| Back | Up | Next |

10.6. Using Jabber with Mon and Nagios

This section provides some tips on how Jabber can be used with system−monitoring tools, such as Mon and Nagios. Jabber alerting is a useful feature for these tools because email alerts do not provide instant notification. Jabber provides a free instant notification anywhere the user is logged into Jabber. This is especially valuable for open source developers who, most likely, are not going to pay for a pager to receive server alerts.

This guide is intended to demonstrate how Jabber can be used with these monitoring tools. See the respective Mon and Nagios sites for instructions on how to use these applications.

10.6.1. Using Jabber with Mon

Mon is a basic service monitoring tool. On startup, Mon reads a configuration file and then performs tests as scheduled in the configuration file. Mon tests consist (primarily) of Perl scripts located in `/etc/mon/mon.d`. If a test fails, Mon calls the alert as specified in the configuration file. Mon alerts are Perl scripts located in `/etc/mon/alert.d`. As of this writing, the Mon distribution comes with neither a Jabberd monitor nor a Jabber alerter; however, the Jabberdoc tools folder contains scripts for both these tasks.

The jabber_alert.pl script was originally written as an alerter for Mon. To use it with mon, simply copy it to the `/etc/mon/alert.d` directory, and chmod it so that it can be run. You may wish to rename it `jabber.alert` to maintain consistency with the other types of alerts.

Once `jabber.alert` is in your `alert.d` directory, you can use it to send Mon alerts. For example, this sample section of a `mon.cf` script would jab `user1@somedomain.org` if the web server for `www.somedomain.com` failed:

```
hostgroup webservers www.somedomain.com

watch webservers
    service HTTP_Server_Check
        interval 1m
        monitor http.monitor −p 80 −t 30
        period wd {Mon−Sun} hr {0am−24pm}
            alert jabber.alert −e user1@somedomain.org −n user2@somedomain.org −w mypasswor
```

The tools folder also contains jabber.monitor, which is a script for monitoring Jabber servers. To use this monitor, copy it to the `/etc/mon/mon.d` directory and chmod it so that it can be run. Its use is similar to that of other Mon monitors. This section of a `mon.cf` script would test the `jabber.somedomain.com` server every minute:

```
hostgroup jabberservers jabber.somedomain.com

watch jabberservers
```

```
service Jabberd_Server_Check
    interval 1m
    monitor jabber.monitor -p 5222
    period wd {Mon-Sun} hr {0am-24pm}
        alert jabber.alert -e user@somedomain.org -n user2.somedomain.org -w mypassword
```

Of course, if you are monitoring a Jabber server, you want neither the Jabber alert sender nor recipient to be using the server being monitored. If you use the same Jabber server for monitoring and alerting, you would receive no alert messages if the server failed. You would want to choose a reliable Jabber server for sending and receiving alerts, and you may wish to peruse the Jabber Monitor Service, which provides graphical uptime statistics for public Jabber servers.

Mon installation and configuration is beyond the scope of this guide; however, I have included a mon.cf.sample configuration file in the tools folder. Note that this configuration relies on the alert and monitor scripts above in addition to a modified dns.monitor script. The modified `dns.monitor` script allows for testing of Jabber DNS SRV records.

A.10.6.2. Using Jabber with Nagios

Nagios offers many more features than Mon, including a web interface that is capable of providing graphical network maps showing problem areas. I am not yet familiar with Nagios; however, there is a Jabber alert script available for it.

| Back | Up |

A.11. Primer on Transports and Jabberd 2

The concept of Jabberd transports can be confusing to new Jabberd 2 administrators. The fact that many Jabberd 2 transports rely on legacy Jabberd 1.4 support makes this especially true for administrators who have no experience with Jabberd 1.4. This appendix provides a primer on Jabberd 2 transports by explaining how transports have evolved:

- Transports and Jabber
- Enter Jabberd 1.4x
- Enter Linked Configuration Files
- Enter Jabberd 2
- Enter JCR
- Additional Information
- Recap

At first glance, this may seem to be more information than a new Jabberd 2 administrator would require; however, understanding legacy transport support in Jabberd 2 requires an understanding of how transports work with the Jabberd 1.4x series server. Note that in this appendix, the term "transport" is applied liberally to mean any external Jabber service.

A.11.1. Transports and Jabber

The Extensible Messaging and Presence Protocol (XMPP): Core draft provides an overview of how transports, known as gateways in XMPP parlance, work with a Jabber server.

Figure A.11.1. Jabber Deployment Diagram (High Level View)*:

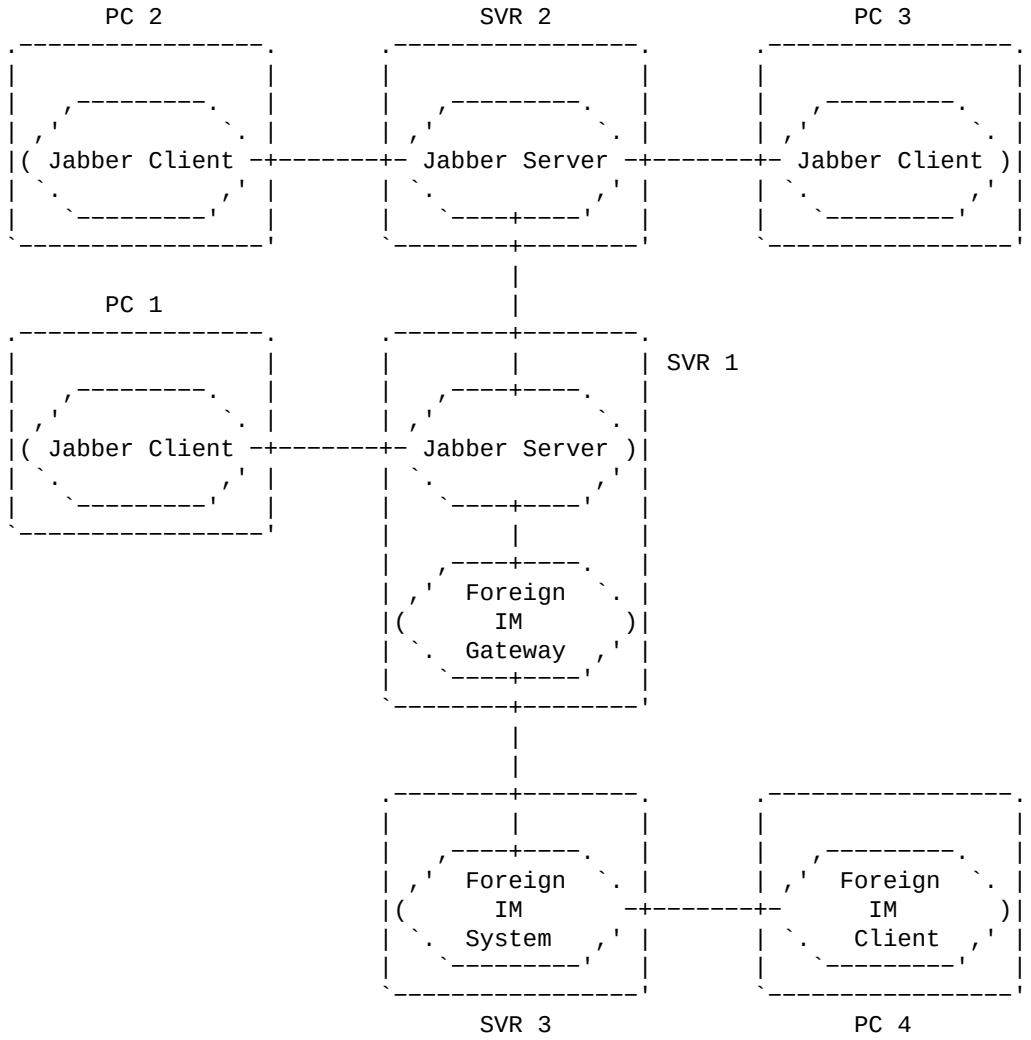
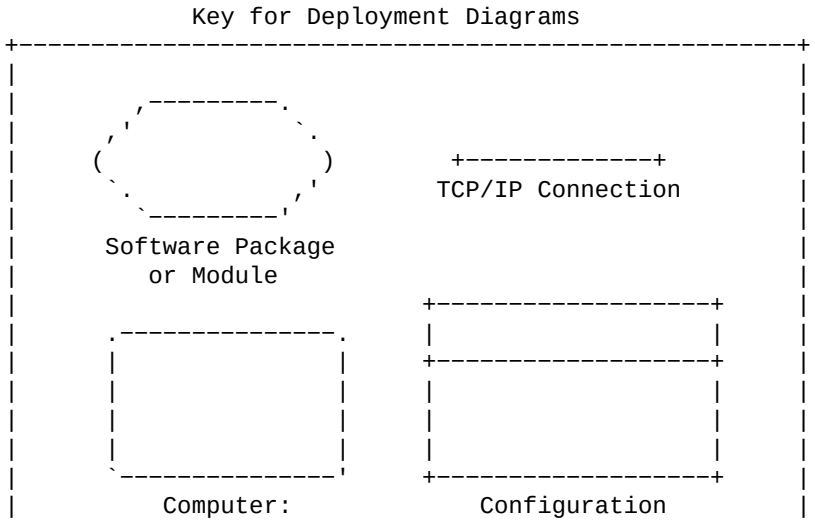


Figure A.11.1. is adapted from XMPP Core Draft

Figure A.11.2. Key to Deployment Diagrams:

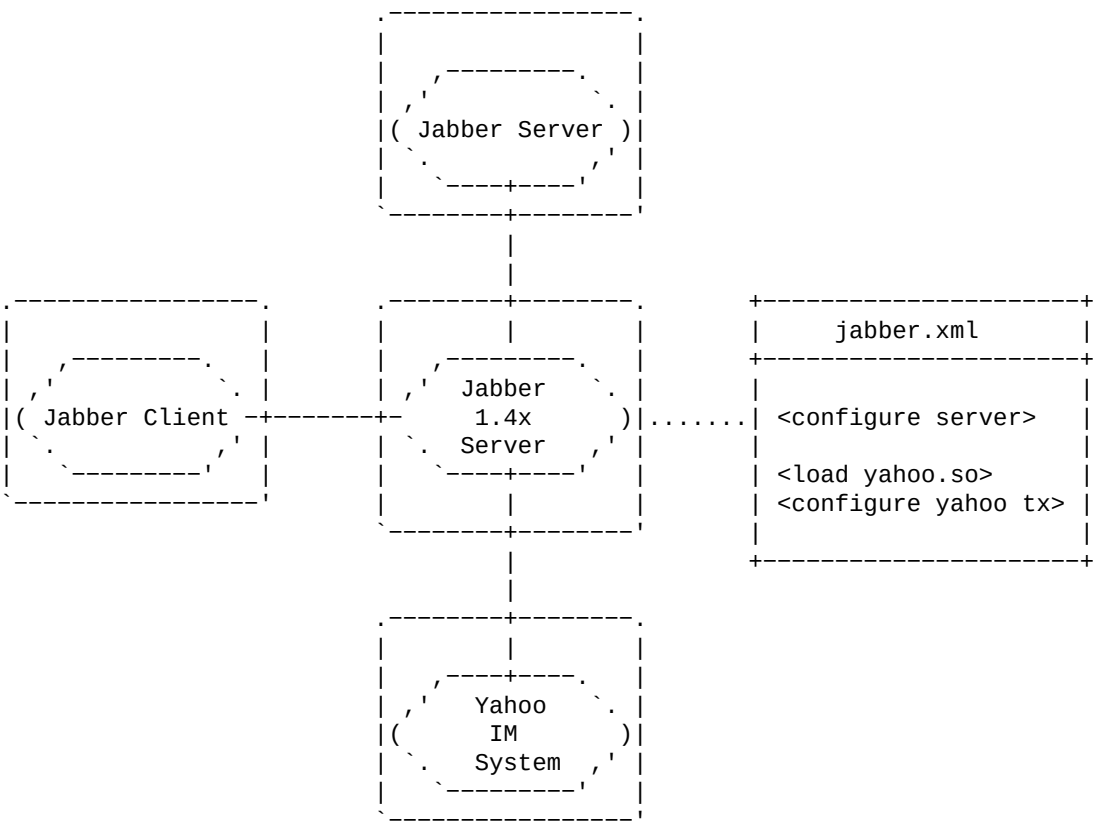


Essentially, a transport is a piece of software, which connects to a Jabber server and which allows client communication with a non−Jabber instant messaging system.

A.11.2. Enter Jabberd 1.4

Transports for the Jabberd 1.4x series server are generally supplied as shared object files that are executed and run by the Jabber 1.4x server. Originally, these shared object files were loaded by the main Jabberd 1.4x process, and configuration for each transport was contained in the main Jabberd 1.4x configuration file. The diagram below shows how a Jabber 1.4x server might load and run a transport called `yahoo`.

Figure A.11.3. Jabberd 1.4x Server Running a Transport Internally:



Note that in these diagrams, the XML configuration files have been extremely simplified. The abbreviation "tx" stands for "transport".

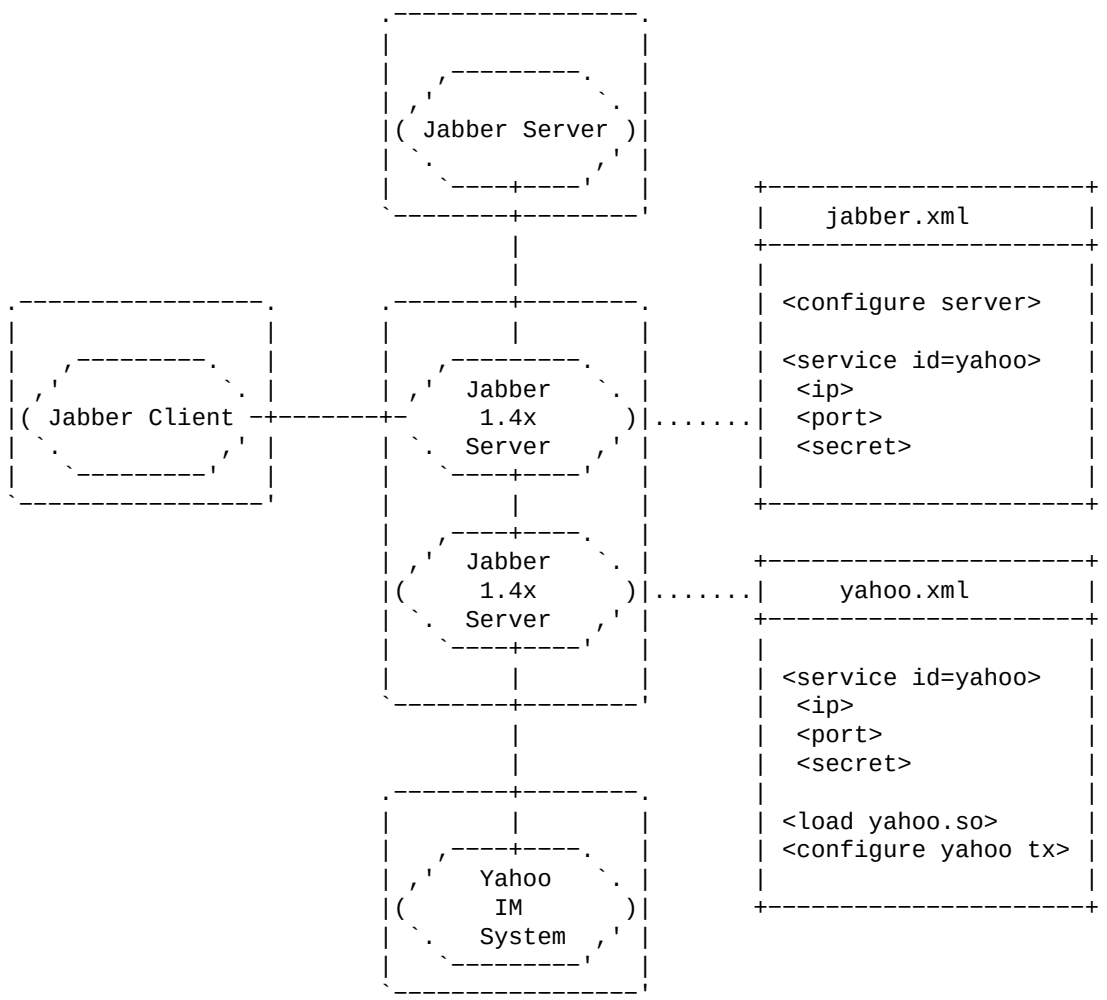
A.11.3. Enter Linked Configuration Files

The problem with the configuration in Figure 11.3. is that the main server process becomes dependent on the transport. That is, a problem with the transport can bring the entire Jabberd 1.4x server down. To address this issue, the 1.4x series Jabberd server supports what I call *linked configuration files*.

The idea behind linked configuration files is that the main Jabberd server does *not* run any of the transports. Rather, each additional transport is run in a separate Jabberd 1.4x process. For each transport, the main Jabberd configuration file contains a section that identifies an external service. Service information includes an ID for the service, an IP address and port to connect to the service, and a shared secret used to authenticate the service.

Each transport exists as a separate Jabberd 1.4x process with its own configuration file. The configuration file for a transport contains the service information found in the main configuration file (ID, address, port and secret) in addition to configuration for the transport itself. Figure A.11.4. continues with the example above and shows how a separate Jabberd 1.4x process could be used to run a transport.

Figure A.11.4. Jabberd 1.4x Server Running a Transport as a Separate Process:



The important thing to note in the above diagram is that the Jabberd 1.4x process running the transport does not make any client connections. All client connections are handled by the main Jabberd 1.4x server. The Jabberd 1.4x process handling the transport connects directly to the main Jabberd 1.4x server.

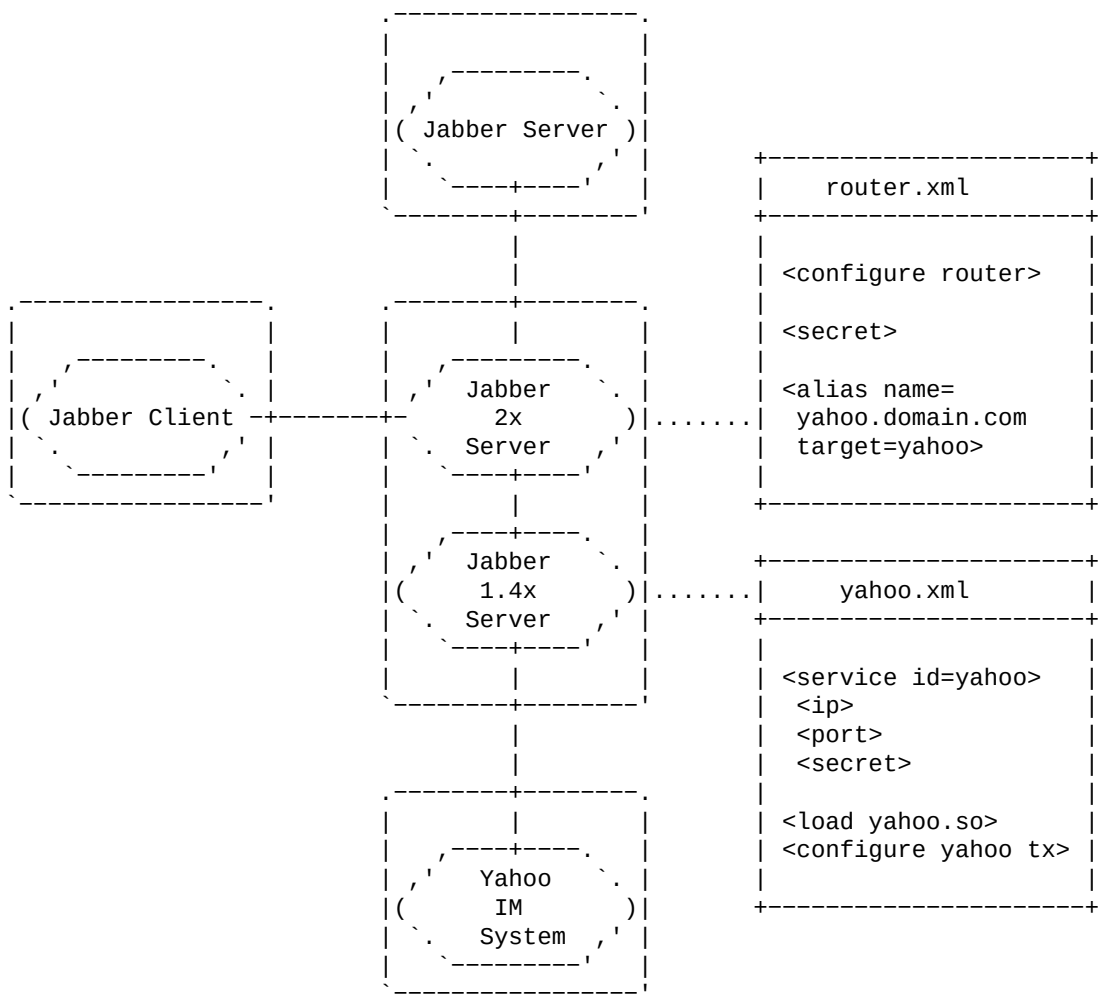
The second thing to notice is that the Jabberd 1.4x process running the transport uses an abbreviated XML configuration file. This configuration file specifies a connection to the main Jabberd 1.4x server, it loads the transport shared object, and it configures the transport.

A.11.4. Enter Jabberd 2

Jabberd 2 provides backwards compatibility for Jabberd 1.4x compatible transports. Jabberd 2 accomplishes this legacy support by supporting transports that are run within a linked Jabberd 1.4x process. Thus, Jabberd 2 can connect to a transport run as in the figure above.

Configuration for a Jabberd 2 transport running within a Jabberd 1.4x process is nearly identical to what the configuration would be if that transport were connecting to a Jabberd 1.4x server. The main differences are that for Jabberd 2, all transports connect to the `router` on port 5347, and all share the same `secret`. Figure A.11.5. shows how a Jabberd 2 server can connect with a transport running within a Jabberd 1.4x process.

Figure A.11.5. Jabberd 2 Server Running a Transport as a Separate Jabberd 1.4x Process:



Note again that the Jabberd 1.4x process running the transport does not connect to clients. It connects with the `router` and therefore does not need access to the client ports of 5222 and 5223. Thus, the transport running on the Jabberd 1.4x process can run side−by−side with the Jabberd 2 server. Moreover, the Jabberd 2 server can run side−by−side with multiple Jabberd 1.4x processes running various transports.

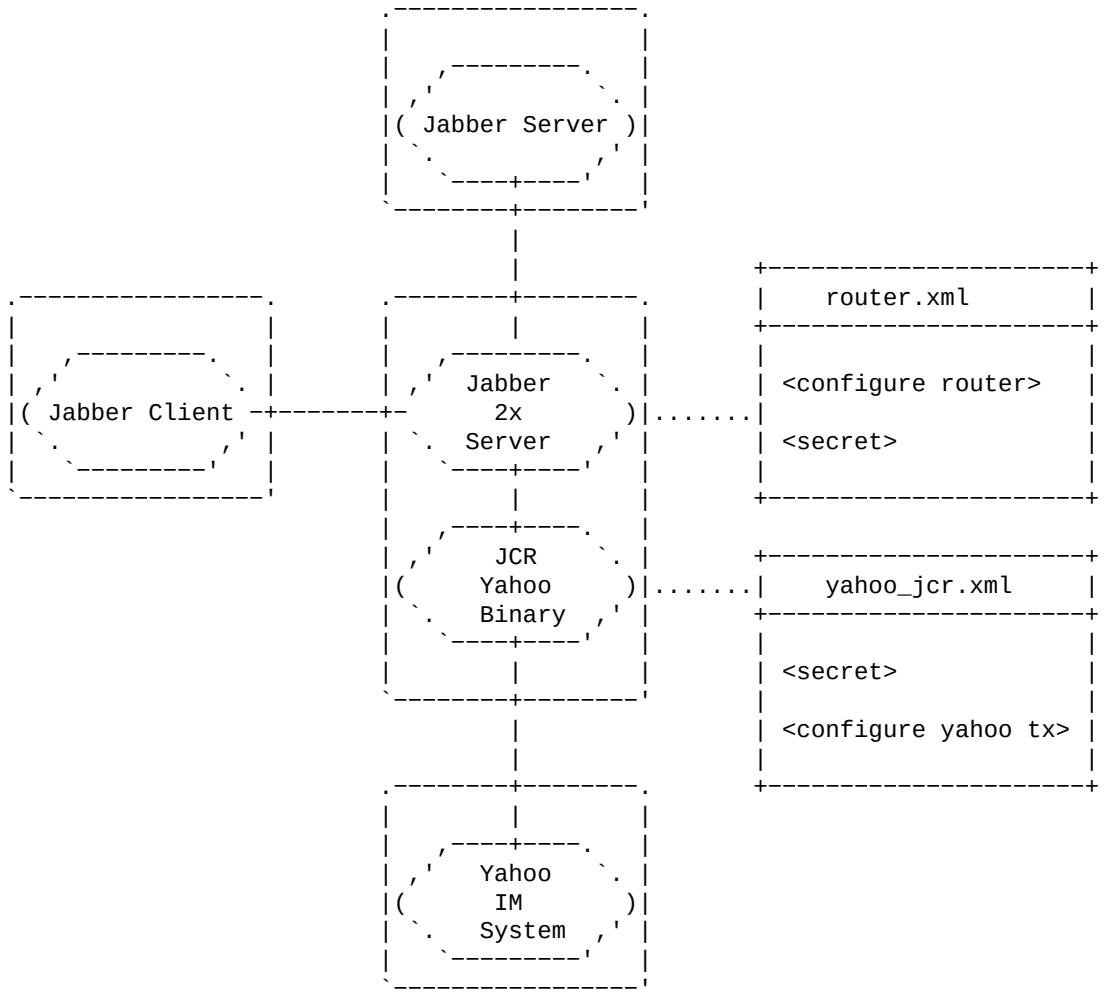
Note also that the Jabberd 2 server does *not* run the transport. The transport is started and run separately in a Jabberd 1.4x process, and it connects to the Jabberd 2 server over a TCP connection.

Section 5.8 of this guide explains how to use Jabberd 1.4x to provide transports for Jabberd 2.

A.11.5. Enter JCR

The Jabber Component Runtime provides a wrapper for Jabber 1.4x compatible transports. Instead of running a legacy transport within a Jabberd 1.4x process, JCR can be used to create a binary file that runs a transport. Essentially, the JCR wrapper provides the services that a separate Jabberd 1.4x process would provide for a transport. Figure A.11.6. shows how Jabberd 2 can connect to a JCR wrapper running a transport.

Figure A.11.6. Jabberd 2 Server Running a Transport Using JCR:



As is true for a transport running within a Jabberd 1.4x process, a transport running within a JCR wrapper runs separately from the Jabberd 2 server. Section 5.9 of this guide explains how to run a transport within a JCR wrapper.

A.11.6. Additional Information

Some transports, especially newer ones, can run natively with the Jabberd 2 server. Python MSN−Transport is an example of a transport that can run natively with Jabberd 2. These newer transports connect to the Jabberd 2 server much in the same way that a JCR−wrapped transport does.

Some older transports do not support discovery. Discovery, or `disco`, is the capability of advertising service availability to clients. Clients see this availability when browsing services on a Jabber server. For a transport that does not support discovery, an entry must be made in `sm.xml` to advertise that transport's availability. See the `Static service list` section of `sm.xml` for an example entry for a transport that does not support discovery.

A.11.7. Recap

To recap the above discussion, administrators have three options for running transports with Jabberd 2:

- Run Transport within Jabberd 1.4x Process
- Run Transport within JCR Wrapper
- Run Transport Natively

Running transports within Jabberd 1.4x processes is the most universal option because at the time of writing, many transports will *only* run within a Jabberd 1.4x process.

A.11.7.1. Run Transport within Jabberd 1.4x Process

Running a transport within a Jabberd 1.4x process means that a Jabberd 1.4x process separate from the Jabberd 2 processes will provide a transport service. To accomplish this, you will need to download, compile and install Jabberd 1.4x, in addition to downloading and compiling the transport.

Instead of running Jabberd 1.4x with a full−blown configuration file, the Jabberd 1.4x process will be run with a minimized configuration file that is designed to load the transport, configure the transport, and connect to the `router` to provide the service. The configuration file for the Jabberd 1.4x process should contain the following elements (in addition to configuration details for the transport itself):

- An XDB section
- A log section
- `router` IP address and port
- `secret` shared by the `router` component (found in `router.xml`)
- An alias for the service (as specified in `router.xml`)

See Section 5.8 for details on how to set up such a configuration file for a Jabberd 1.4x process running a transport. Note that many transports contain an example for a *linked configuration file*. Such an example forms the basis for a Jabberd 1.4x configuration file that can be used to run a transport under Jabberd 1.4x with Jabberd 2.

A.11.7.2. Run Transport within JCR Wrapper

Running a transport within the JCR wrapper means that a dedicated binary will run the transport. To accomplish this, you will need to download JCR and the transport. Following the instructions included with JCR, you will first build JCR and then compile it against the transport to create a binary that will run the transport. The JCR binary will then be run as a separate process with its own configuration file, and this configuration file will specify connection with the `router` component, including the shared `secret`. See Section 5.9 and the JCR Home Page for more information about running a transport with JCR.

A.11.7.3. Run Transport Natively

Running a transport natively means that the transport is run as its own standalone process with its own configuration file. A natively−run transport will connect with the `router` component of Jabberd 2, and thus, the configuration file must contain connection details for the `router`, including the shared `secret`.

Up

A.12. Troubleshooting Tips for Jabberd 2

This section is designed to provide some troubleshooting tips for Jabberd 2. It is not intended as an exhaustive diagnostics or troubleshooting tree. Additional tips are extremely welcome, especially because I know that there are several regular Jadmin and Jabberd list posters who are more adept at troubleshooting than I am. Please post any additional tips you may have, and I will incorporate them into this page.

General

- Enable debugging
- Go over configuration files with a fine tooth comb

Problems Building Jabberd 2

- Make sure all dependencies are installed
- Specify all options and paths
- Check config.log

Jabberd 2 Crashes After Start

- Make sure all Jabberd 2 binaries have stopped
- Be careful when running as root
- Make sure jabber user has path to binaries
- Make sure that jabber user has access to necessary paths
- Enable debugging
- Go over configuration files with a fine tooth comb

Clients Cannot Connect to Server

- Check DNS resolution
- Make sure user name matches server
- Enable debugging
- Go over configuration files with a finetooth comb
- Use client debug XML output
- Try a different client

General

Enable Debugging

If your Jabberd 2 server does not start, or it has problems connecting to clients, rebuild Jabberd 2 with the `−−enable−debug` switch, and then start it with the `−D` option. The amount of debug output

may be a bit daunting to a new administrator; however, there are a few things to look for.
A Jabberd 2 server crash is usually preceded by the crashing of one of the Jabberd 2 components (`router`, `resolver`, `c2s`, `s2s` or `sm`). Near the end of the debug output, you should be able to figure out which component crashed. Then, you can work backwards in the debug output looking for output for the failed component to see what its last actions were before crashing. This may provide some clues as to why the component crashed. Note that crashes of `sm` and `c2s` are often related to database and database−connectivity issues.
You can pipe the debug output if you find it hard to read from a terminal. For example, you might pipe debug output to a file so that you can search it:

```
/usr/local/bin/jabberd −D > /tmp/jabber_debug.log
```

An alternative is to pipe the debug output through `grep` so that you only see certain messages. This works especially well for monitoring a single component. This command would run Jabberd 2 in debug mode, while showing only debug messages from the `sm` component:

```
/usr/local/bin/jabberd −D | grep ^SM
```

If your server has problems connecting to clients, start Jabberd 2 in debug mode and wait for all components to fully initialize before attempting to connect from a client. Then, view the Jabberd 2 debug output as you attempt to connect from a client.
If you see no additional debug output as your client attempts to connect, you most likely have a DNS, user name or network/firewall problem. If you do see additional debug output as the client connects, look especially for output related to authorization problems.
If you still are unable to troubleshoot your server, you may wish to post a question along with debug output to the jadmin or jabberd lists

Go over configuration files with a fine tooth comb

Many Jabberd 2 issues are caused by a configuration mistake or typo. Go over your configuration files very carefully. You may wish to `diff` each configuration file against the template to see what changes you have made. For example, you might `diff` your `c2s.xml` file to verify changes you have made:

```
cd /usr/local/etc/jabberd
diff c2s.xml c2s.xml.dist
```

If your setup is new, you may wish to consider deleting any modified configuration files and starting over by copying the `*.dist` files in `/usr/local/etc/jabberd`.

Problems Building Jabberd 2

Make sure all dependencies are installed

See Section 2.4 for a description of dependencies for Jabberd 2. If you are using MySQL with Jabberd 2, make sure that each of the Server; Client Programs; Libraries and Header files; and Dynamic Client Libraries packages are installed.

Specify all options and paths

The `configure` script for Jabberd 2 does not necessarily halt if it is unable to find a specific package, nor does it report invalid configure options. The best way to ensure that all packages are found is to specify all packages to use, along with their include and library paths when running `configure`. For example, you might use a `configure` statement like this to build Jabberd 2 and use MySQL for authentication and storage after verifying that the extra include and library paths covered all the dependencies:

```
./configure −−enable−mysql −−enable−ssl −−enable−idn \
−−with−extra−include−path=/usr/include/mysql:/usr/include \
−−with−extra−library−path=/usr/lib/mysql:/usr/lib
```

The key is to identify all the correct include and library paths. `ldconfig` is a useful tool for finding library files. For example, this command would report the locations of MySQL library files (as super−user):

```
ldconfig −p | grep mysql
```

The good thing about using `ldconfig` is that it reports libraries currently being used. This is especially useful if you have performed multiple installations of a package. Once the libraries for a dependency are found, the include files can usually be found in a parallel include directory. For example, if the libidn libraries were found in `/usr/local/lib`, the headers would most likely be found in `/usr/local/include`. If `ldconfig` fails to find a set of libraries, you may need to run it without the `−p` switch (see the man page).

Identifying include and library paths is especially important on Redhat and Fedora because these distributions install packages to non−standard locations. See the FAQ for Redhat and Fedora configuration issues.

Check config.log

The `./configure` output is written to `config.log`. This is a huge file; however, you may be able to do some searching to identify problems. Searching for the whole word "no" is one way to find files that were not found when `./configure` was run. Note that `config.log` reports all files not found as it searches multiple locations. Therefore, each unfound file is not necessarily critical.

Jabberd 2 Crashes after Start

Make sure all Jabberd 2 binaries have stopped

When Jabberd 2 crashes, one or more of the components (`router`, `resolver`, `c2s`, `s2s` and `sm`) may remain in a running state. Restarting Jabberd 2 while a component is running will only cause another crash. Before restarting, make sure that all Jabberd 2 components have stopped. Manually kill any running components before a restart. If you are debugging Jabberd 2, you may wish to use a simple shell script that will kill all Jabberd 2 processes:

```
#!/bin/sh
killall router
killall resolver
killall c2s
killall s2s
killall sm
```

Be careful when running as root

You should not run Jabberd 2 as root. Be aware that running Jabberd 2 as root may cause you additional problems. This is especially true if your installation uses Berkeley DB. If the Berkeley DB database is initialized by the root user, your `jabber` user will not have permissions to write to it.

Make sure that jabber user has path to binaries

Make sure that your `jabber` user is able to access a path to the Jabberd 2 binaries if you `su` to your `jabber` user in a directory where the `jabber` user has no permissions, the `jabber` user will not be able to run Jabberd 2. You may wish to `cd` to the root directory before you `su` to your `jabber` user.

Make sure that jabber user has access to necessary paths

See Section 2.3 for information about creating directories and setting permissions for PID files and logging. By default, Jabberd 2 writes PID files to `/usr/local/var/jabberd/pid`, and logging is done to `syslog`. If you change logging to log to files, make sure that your `jabber` user has write permissions to the log directory.

Clients Cannot Connect to Server

Check DNS resolution

Check DNS resolution for your Jabberd 2 `id`. Your client must be able to resolve the `id` as set in `sm.xml` (near the top of the file) and in `c2s.xml` (under the `local` section):

```
dig jabber.somedomain.com
```

If you are testing Jabberd 2 locally, you may wish to set the host in `/etc/hosts` (in `$WINDOWS\system32\drivers\etc\hosts` or `$WINDOWS\lmhosts` for Windows).

Make sure user name matches server

Make sure that your client is set to connect as a user that matches your Jabberd 2 `id` (as set in `sm.xml` and `c2s.xml`). For example, if your Jabberd 2 `id` is set to `jabber.somedomain.com`, your users must connect as something like `some_user@jabber.somedomain.com`.

Use client debug XML output

Use the debug output from your Jabber client, in addition to the debug output from Jabberd 2. Many Jabber clients provide very limited information for failed connections. To remedy this, enable the debug output in your Jabber client and attempt to connect to your Jabberd 2 server.

Try a different client

Try using a different Jabber client. You may wish to try this simply to rule out any potential Jabber client issues.

Up

Jabberd 2 FAQ

General

Client−Side Issues

Building and Installation

Database Specific

System Specific

General

Q. I've read the guide and the FAQ; however, I still have a question. Where can I search for answers?

A. There are several good sources for answers. The Jabber FAQ's cover general Jabber questions:

- ◊ Jabber General FAQ
- ◊ Jabber End User FAQ
- ◊ Jabber Development FAQ

You can also try searching the Jadmin Archive for questions about jabber administration, or subscribe to the Jadmin Mailing List to post your own question. You may also want to check out Troubleshooting Tips for Jabberd 2.

Q. Where is the Jabberd 2 homepage?

A. Jabberd 2 Homepage

Q. There's a feature that I'd like to see included with Jabberd 2. Where should I post my suggestion?

A. You can post a feature request to Jabberd 2 Feature Requests

Q. I think I've found a bug in Jabberd 2. Where should I post this information?

A. If you are confident that you've found a bug, you can post it to the Jabberd 2 Bug List, or you can post a question to the Jabberd List.

Q. How does Jabberd 2 handle multiple domains?

A. You can add multiple ID's to the Client to Server (C2S) configuration; however, you have to setup individual instances of the session manager (SM) for each domain.

Q. When will components be available for Jabberd 2?

A: Existing components can all be made to work with Jabberd 2. Jabberd 2 does not have an internal component API, so shared object components are impossible.

Q. How do I get existing components to work with Jabberd 2?

A. "External" components (ie components that connect to a Jabber server over TCP using the "jabber:component:accept" protocol will work as−is. Simply configure the component to connect to the router using the port and secret specified in router.xml.

"Internal" or "library load" components that load at runtime into a jabberd 1.4 instance can be used by placing the 1.4 instance into "uplink" mode. An appropriate "linker" alias needs to be added to router.xml to support this.

Jabberd 2 does not provide XDB and logging facilities. For "internal" components, these can be provided by the enclosing jabberd 1.4 instance. For "external" components, they will need to connect to a 1.4 instance that provides these services, which in turn uplinks to the Jabberd 2 router.

Client−Side Issues

Q. I just created a user in the database. Why can't I login as the user I just created?

A. It is not enough to add users to the `authreg` table because this only introduces users to the c2s component, but not to the sm component. Correct entries are required in the `active` table as well. It is best to use a Jabber client to register users. If this is not acceptable, corresponding entries can be manually created in the `active` table, or the session manager can be configured to create new users automatically the first time they log in. Uncomment the `auto−create` tag in the `User options` section of `sm.xml` to enable auto−creation of new users. (Note that enabling `auto−create` does not enable inband registration, as only users that pass the c2s component are created.)

Building and Installation

Q. Why does configuration fail to find my MySQL installation?

A. The most common reason why Jabberd cannot find the MySQL libraries is that not all of the required libraries are installed. In addition to the basic MySQL installation, Jabberd requires that the development libraries and headers be installed. Either perform a *Max* installation as listed on the MySQL Downloads page, or install *Server, Client Programs, Libraries and header files*, and *Dynamic client libraries* separately.

Q. Where can I find packages (RPM's, Debian, etc.) for Jabberd 2?

A. Debian packages for Jabberd 2 are now available (see rpmseek.com). RPM's are not available at this time.

Q. When configuring Jabberd 2 stable 3, why does configure say it cannot find Libidn?

A. Starting with Jabberd 2 Stable 3, Libidn no long ships with the Jabberd 2 distribution. This is due to a licensing conflict. Users must install Libidn themselves. See Section 2.4.2 and Appendix 5.

Q. When configuring Jabberd 2 Stable 3, why can't configure find MySQL (or other package) that worked fine with a previous release?

A. The configure script and options have changed with Jabberd 2 stable 3. The `−−with−package=path` has been replaced with `−−with−extra−include−path path` and `−−with−extra−library−path path`. The paths searched also changed with stable 3;

therefore, it may be necessary to specify a path(s) to a package that was automatically found by a previous release. See Section 3.3.

Q. Why does Jabberd 2 stable 3 crash after a successful configure and build?

A. The configure script and options have changed with Jabberd 2 stable 3. The `--enable-authreg=package` and `--enable-storage=package` have been replaced with a simpler `--enable-package` option. Thus to enable MySQL for authreg and/or storage, one would use the option `--enable-mysql`. See Section 3.3. Because the configure script does not report invalid options, using an option such as `--enable-authreg=mysql` would cause neither the configure nor the build to crash; however, the MySQL driver would not be compiled in. If you then set the XML configuration files to use MySQL, Jabberd 2 would crash.

Database Specific

PostgreSQL

Q. Why do login and user-create fail on my installation using PostgreSQL?

A. In order for PostgreSQL to work with Jabberd 2, PostgreSQL must be setup to accept TCP/IP connections. Verify that PostgreSQL is listening on an inet socket (`netstat -l |grep postgres`). If not, modify `postgresql.conf`, setting `tcpip_socket = true` and restart. If problems still exist, check the `pg_hba.conf file` and check if TCP/IP connections are authorized (i.e. an appropriate host record exists).

Berkeley DB

Q. Why does ./configure fail to find my Berkeley DB installation?

A. The `./configure` script searches several locations; however, if the script cannot find your Berkeley DB installation, you should specify the location as a parameter when running `./configure`. For example, if you are running Berkeley DB version 4.2, you may need to run `./configure` as below:

```
./configure --with-berkeley-db=/usr/local/BerkeleyDB.4.2/
```

MySQL

Q. When working in the MySQL console, why can't I query table names that are hyphenated?

A. In MySQL statements, you need to enclose hyphenated names with back ticks as shown below:

```
select `collection-owner`, `object-sequence`, `jid`, `group` from `roster-groups
```

Q. I just installed Jabberd 2 stable 3. Why do I get errors stating that the database connection to MySQL failed?

A. Jabberd 2 stable 3 connects to the MySQL server socket at `/tmp/mysql.sock`. The default socket when installing MySQL from source is `/var/lib/mysql/mysql.sock`. You will need to create a symlink to `/tmp/mysql.sock` if it does not exist:

```
ln -s /var/lib/mysql/mysql.sock /tmp/mysql.sock
```

System Specific

Redhat

Q. On Redhat, why does ./configure fail to find MySQL?

A. The most likely reason for `./configure` to fail on Redhat is that Redhat ships with a MySQL installation that does not include the development libraries required by Jabberd 2. See the MySQL question under *Building and Installation* above. The second reason that `./configure` might fail is that Redhat stores the MySQL libraries and include files separately. Run `./configure` with both paths (as below) to identify both locations:

```
./configure --with-mysql=/usr/include/mysql:/usr/lib/mysql
```

Q. On Redhat 9, why does the build crash after a successful configure?

A. The most likely reason for the build to crash on Redhat 9 is that RH 9 ships with its own version of Kerberos. The OpenSSL libraries are found when `./configure` is run. However, Kerberos references cause the Jabberd 2 build to crash.

There is an easy work around for this problem. Execute the command below before running `./configure`:

```
export CFLAGS="-I/usr/kerberos/include"
```

Q. On Redhat, why does ./configure fail to find Berkeley DB even when I specify a path to it?

A. If `./configure` cannot find Berkeley DB, it may be because libpthreads is not linked. Run `./configure` as below to fix this:

```
./configure [options] LDFLAGS=-lpthread
```

Mandrake

Q. On Mandrake, why does ./configure fail with the error that mysql (or pgsql)cannot be found?

A. Either the database include files or library files cannot be found by `./configure`. To fix this, run `./configure` with the paths to both as below (replacing "mysql" with "pgsql" for PostgreSQL installations):

```
./configure --with-mysql=/usr/include/mysql:/usr/lib/mysql
```

Debian

Q. On a clean Debian install, why does ./configure fail to find MySQL?

A. On Debian, zlib must be installed.

Up