How To Use XMPP With BigWorld

BigWorld Technology 2.0. Released 2010.

Software designed and built in Australia by BigWorld.

Level 2, Wentworth Park Grandstand, Wattle St
Glebe NSW 2037, Australia
www.bigworldtech.com

Table of Contents

Chapter 1. Overview

XMPP is an open XML based message protocol with origins in Instant Messaging (IM) and friend presence notification. It is presented within the scope of the BigWorld FantasyDemo example code to illustrate a working implementation of the `BigWorld.registerFileDescriptor()` method as well as providing a real world example of one approach that can be used for implementing a messaging system into your game using an external service.

The intended audience for this document includes technical MMOG developers, designers and system administrators.

XMPP was chosen as the protocol to use for illustrating in game messaging services for a number of reasons:

- Open Standard

  XMPP provides a completely open protocol standard that can be re-implemented by anybody without any licensing costs. This also means that there are many free implementations of the XMPP protocol available.

- Decentralised architecture

  The decentralised nature of the XMPP protocol lends itself to server solutions that can scale as required and integrate with other services.

- Integration with other services

  Protocol extensions have been defined that describe how other services can interact with an XMPP server through various mechanisms. This allows users on an existing service such as AOL or ICQ to communicate with users on an XMPP server as though they are all within the same network.

XMPP has been used in numerous projects including, but not limited to:

- Quake Live

  Source: Process One Blog

- Google Wave

  Source: Google Wave Federation Protocol

- Jabber IM

  Source: Jabber.org

- Gtalk

  Source: GTalk Developer Documentation

There are numerous XMPP server implementations available including both free and commercial solutions. For the purposes of this document we use EjabberD (http://www.process-one.net/en/ejabberd/).

1.1. Free / Open Gateways

Currently the only known open and free to use XMPP gateway is the GTalk gateway operated by Google. Refer to the federation documentation of your XMPP server for more details on integration.

Chapter 2. Installation

2.1. Installing the XMPP server

For the purposes of this example we use the ejabberd XMPP server. This is available on all BigWorld supported distributions using package management, however if necessary the official website can be found at:

http://www.process-one.net/en/ejabberd

To install ejabberd under CentOS or Red Hat Enterprise simply perform the following step as the root user:

```
# yum install ejabberd
```

Note

Red Hat Enterprise and CentOS users will need to enable the Extra Packages for Enterprise Linux (EPEL) package repository to gain access to this.

2.2. Configuring the FantasyDemo example

The IM chat integration example needs to know how to connect to the newly installed Jabber server. The script code reads its configuration from `fantasydemo/res/server/config/xmpp.xml`. In this file you will need to specify the details of the XMPP server (i.e. hostname and port) as well as a resource name. The resource name is a unique identifier that specifies to the server where a user is logging in from when a connection is made. The default resource name is "FantasyDemo", however this can be altered to your needs as required.

In this file also ensure that the example has been enabled with the corresponding XML tag.

```
<root>

    <enabled> true </enabled>

    <!-- XMPP Chat example -->
    <xmppServer>
        <host> ejabber_server.domain.com </host>
        <port> 5222 </port>
        <resourceName> FantasyDemo </resourceName>
    </xmppServer>
</root>
```

2.3. Configuring EjabberD

Most package installations of ejabberd will place the configuration file into `/etc/ejabberd/ejabberd.cfg`. When discussing the ejabberd config file, this is the file that should be referenced.

Note

When modifying or adding entries to the configuration file, the entries are terminated with a full stop '.'.

2.3.1. Servicing Domains

The ejabberd server services a domain of logins. For example your environment may contain both `www.rpg-game.com` as well as `www.rts-game.com`. Each gaming environment could contain a unique user called 'tony', so we need to tell the ejabberd server that it has to service both these domains.

By default most installations will have a working configuration for `localhost`, however we will extend this to also support a new servicing domain so other users in the office can login to the server and interact with the game clients. In order to do this we simply use the fully qualified host and domain name of the machine the ejabberd server has been installed on. For example, while testing during the writing of this installation document the ejabberd server was installed onto a host called `bw13`, so the fully domain being serviced would be `bw13.bigworldtech.com`. In the ejabberd config file edit the hostname line to look similar to the following, replacing the bw13 domain with your own hostname and domain name.

```
{hosts, ["localhost", "bw13.bigworldtech.com"]}.
```

2.3.2. Allowing multiple account creations from a single IP

By default the ejabberd server will limit the number of account registrations that are allowed to occur from the same IP address in order to stop the server being abused. As the ejabberd server will be running in an isolated environment where account creations can happen frequently from the same IP address (i.e. the same BaseApp registering multiple new accounts at once), we will disable this limitation. To do so, add the following entry to the ejabberd configuration file:

```
{registration_timeout, infinity}.
```

2.4. Start the ejabberd server

Now that the server has been configured start the server. On most distributions this can be achieved using the following command run as the root user:

```
# /etc/init.d/ejabberd start
```

Chapter 3. Usage

3.1. Connecting to ejabberd using a client

Once the ejabberd server is running, it is now possible to connect to it using a standard instant messaging client such as Gajim, Pidgin, or any other of the myriad of clients that support XMPP. Please be aware that we have had reports of the Trillian Astra Pro IM client not being able to create accounts with the ejabberd setup described in this document.

Connecting using a normal IM client will allow you to confirm that you have correctly configured the ejabber server before attempting to connect with a BigWorld server and client.

Your client will be able to create a new account when attempting to connect. If you are prompted for a "Resource name", the name of the IM client you are using will suffice.

If you have trouble connecting at this stage it is most likely either a mis-configuration of the ejabber server or an incorrectly entered set of details when trying to connect with the client. To check the server configuration you can check the ejabber server logs which will should be located for most distributions under `/var/log/ejabberd`.

3.2. Starting a Fantasy Demo Server

Provided the ejabber installation was performed correctly and the server details were correctly provided, simply starting a server normally should be enough to enable the example. All the initialisation of the example is performed through the BaseApp personality script `FantasyDemo.py`.

To confirm that the Chat example has been enabled, search the BaseApp `SCRIPT` output for the following line:

```
XMPP Chat example enabled
```

3.3. Connecting using the FantasyDemo client

On a Windows machine simply start the Fantasy Demo client as normal and connect to the server that was started in “Starting a Fantasy Demo Server”on page 9. Once connected and in the game world there are 4 operations that can be performed:

- Add a friend to the players friends list
- Message a friend
- List all friends in the friends list
- Delete a friend from the friends list

Note

For all the commands that require a `<friend name>`, in order for the Fantasy Demo script to differentiate between the in game friend lists and the XMPP friends list, the friend name MUST be a JID (which looks like an email address) e.g. `friend@domain.com`. The domain name will be whatever was specified when you configured your ejabber server in “Servicing Domains” on page 8 .

- To add a friend

```
/addFriend <friend name>
```

- To message a friend

```
/msgFriend <friend name> : Message to your friend
```

- To list all your friends

```
/listFriends
```

- To delete a friend from your friends list

```
/delFriend <friend name>
```

3.4. Communicating to the game client from your IM client

Now you should have a connected game client and a connected IM client , you should be able to communicate between the two. If you haven't already, add your game client avatar name to your IM clients friend list e.g. `avatar_name@domain.com`, and ensure that you have added the IM client friend to your avatar's friends list using `/addFriend im_name@domain.com`.

You should now be able to send messages between the two.

3.5. Communicating to the game server from your IM client

FantasyDemo also has an `XMPPEventNotifier` entity which you can communicate with via your IM client. This entity broadcasts a text message whenever a Player logs on or off and can respond to watcher queries. XMPPEventNotifier's JID is `server_of_<username>@<jabber server>`, where <username> is the name of the user running the server.

- To query a watcher, send the message

```
getWatcher <watcher path>
```

Chapter 4. Implementation

This chapter outlines the important components of BigWorld's XMPP example that is included with FantasyDemo. It is not intended as a complete discussion of the implementation, more of a guide to assist you in working through the parts of the code most relevant to your needs.

Python source code relevant to the FantasyDemo example can be found under the `fantasydemo/res/scripts/` directory. Specific files are outlined below in more detail as each component is discussed.

The BigWorld XMPP example can be broken down into four components that are worth discussing individually.

- XMPP Module
- XMPP Roster
- Avatar Implementation
- XMPP Event Notifier

4.1. XMPP Module

Location: `fantasydemo/res/scripts/base/xmpp/`

At the time of implementing the XMPP example there were no Python XMPP modules available with a suitable license for redistributing with BigWorld. As a result BigWorld implemented an small module for communicating with an XMPP server which can be found in the afore mentioned directory.

4.1.1. xmpp.Connection

This module class implements the basic communication with an XMPP server. It is responsible for talking XMPP over the wire and communicating the results via the `xmpp.Connection.ConnectionHandler` interface.

4.1.2. xmpp.Client

Any object that wishes to communicate with an XMPP server can inherit from `xmpp.Client` and implement concrete versions of the `onXmpp*` methods to receive notifications from the server in response to roster events.

This implementation also passes through a connection handler (see `xmpp.Connection.ConnectionHandler`) in `xmppConnect` rather than implementing the methods itself to allow each user of the XMPP client to implement its own connection based functionality.

4.1.3. xmpp.Parser

This class offers basic XML parsing to separate data received from the XMPP server into useable pieces. This class is only used by the `xmpp.Connection` class as part of its communication with the XMPP server.

4.1.4. xmpp.Stanzas

This file provides a set of template XML blocks that are used by the `xmpp.Connection` object for communicating with the XMPP server.

4.1.5. xmpp.Service

`xmpp.Service` implements a simple initialisation routine to test that an XMPP server that has been specified in the configuration file exists and is ready to communicate.

4.2. XMPP Roster

Location: `fantasydemo/res/scripts/common/XMPPRoster.py`

The `XMPPRoster` class offers a generic storage location for friends that exist in a users XMPP roster[1].

The XMPPRoster specifically handles the possibility that multiple transports are in use (eg: XMPP / MSN / ICQ) and allows querying for friends across all transports.

4.3. Avatar Implementation

Location: `fantasydemo/res/scripts/base/Avatar.py`, `fantasydemo/res/scripts/client/Avatar.py`, `fantasydemo/res/scripts/entity_defs/interfaces/XMPPClient.def`

4.3.1. Base Implementation

The Base Avatar class simply inherits from `xmpp.Client` and implements `xmpp.Connection.ConnectionHandler` interface which is then provided to `xmppConnect`.

The only point of note with the usage of `xmpp.Client` is that because of the `xmpp.Connection` socket, it is not possible to backup `xmpp.Client.xmppConnection`. Thus if an entity that implements `xmpp.Client` is restored, it needs to re-establish the XMPP client connection in the `onRestore` method.

4.3.2. XMPPClient Interface

This interface is provided as a simple way to pass through Client calls to the `xmpp.Client` methods that the Avatar base entity has available due to its inheritance.

4.4. XMPP Event Notifier

Location: `fantasydemo/res/scripts/base/XMPPEventNotifier.py`

The XMPP Event Notifier is very similar to the Avatar example, in that it uses an `xmpp.Client` to communicate with the XMPP server, however it does not have the same need to inherit from the `xmpp.Client` as it is not a Proxy and will not need the XMPPClient interface.

[1] The XMPP specification refers to a friends list as a roster to generalise the concept of a collection of users.