Introduction

Instant messaging (IM) communication is done through the use Extensible Messaging and Presence Protocol (XMPP) (also known as Jabber), an open standard XML based messaging communication protocol.

Most IM Clients use the XMPP protocol for communication. Google is one of the clients that use XMPP for its Google Talk Product.

There are various third party libraries available for developing Instant Messaging client applications. One of the most popular XMPP client libraries is Smack API, which is a pure Java Library which allows developers to create IM Clients. Following is a walkthrough of building a simple chat application using the Smack API.

Before starting to develop and explore some of the features of Smack API, developers are requested to go through the technical documentation and API documentation of Smack API (3.2.2 latest version at this time of development of this article)

SmackAPI

The Smack API is a pure Java Library. The asmack third party library has heavily patched the Smack API to work with Android. For more information, visit the Smack API community thread.
Libraries are available under the following section:

- download -- asmack

Download the asmack library jar file with naming convention asmack-<date>.jar, where <date> is the date of release. Add it to your project lib folder.

The sample chat application does the following:

- Connects to a GTalk Server
- Logs into GTalk Server
- Sets user Presence
- Gets Rosters
- Sends Messages
- Receives Messages

Connecting to a Server

In this document, Google Talk Server is used for the development of sample chat app.

Connecting to XMPP server requires knowing of configuration parameters set by XMPP Server.

Following are the required configuration parameters for connecting to GTalk Server:

```
public static final String HOST = "talk.google.com";
public static final int PORT = 5222;
public static final String SERVICE = "gmail.com";
```

[Code 1]

For more information on configuration parameters, visit GTalk Developers site.
Following code snippet shows how to use the Smack API for connecting to XMPP Server

```
ConnectionConfiguration connConfig = new ConnectionConfiguration(HOST, PORT, SERVICE);
XMPPConnection connection = new XMPPConnection(connConfig);

try {
  //Connect to the server
  connection.connect();
} catch (XMPPException ex) {
  connection = null;
  //Unable to connect to server
}
```

[Code 2]

The XMPPConnection class is used to create the connection to the XMPP server specified by the ConnectionConfiguration class, which uses configuration parameters for establishing connection with the server.

To disconnect, use the disconnect() method.

Login to a Server

Once a connection is established, the user should log in with username and password using the login() method of the Connection class. Following code snippet shows how to login:

```
//Most servers require you to login before performing other tasks.
connection.login(USERNAME, PASSWORD);
//for example: connection.login(“abc@gmail.com”, “password”);
```

[Code 3]

Once logged in, user can start using chat features; Chat or GroupChats, Set Presence, Get Rosters etc.

Setting user Presence

After logging in, the user might set his or her presence (availability) status visible to other recipients (Rosters) present in the chat list.

The following code shows how to set the presence. The Presence object is created with a status set to type “unavailable”. This presence status is later send as packet using the Connection class’s sendPacket() method.

```
// Create a new presence. Pass in false to indicate we're unavailable.
Presence presence = new Presence(Presence.Type.unavailable);
presence.setStatus("I’m unavailable");
connection.sendPacket(presence);
```

[Code 4]

Getting Roster

The Roster class does the following

- •Keeps track of the availability (presence) of other users
- •Allows users to be organized into groups such as "Friends" and "Co-workers"
- •Finds all roster entries and groups they belong to
- •Retrieves the presence status of each user.

Retrieving the roster is done using the Connection.getRoster() method.

```
Roster roster = connection.getRoster();
//Get all rosters
Collection<RosterEntry> entries = roster.getEntries();
//loop through
for (RosterEntry entry : entries) {
//example: get presence, type, mode, status
Presence entryPresence = roster.getPresence(entry.getUser());
  Presence.Type userType = entryPresence.getType();
  Presence.Mode mode = entryPresence.getMode();
  String status = entryPresence.getStatus();
}
```

[Code 5]

The previous code provides the status of users at a given point in time, but to obtain users' Presence in real time, RosterListener (interface) is used. The Callback methods are called whenever there is change in roster or change in the presence of users in the roster.

```
roster.addRosterListener(new RosterListener() {
 @Override
 public void presenceChanged(Presence presence) {
   //Called when the presence of a roster entry is changed
 }
 @Override
 public void entriesUpdated(Collection<String> arg0) {
   // Called when a roster entries are updated.
 }
 @Override
 public void entriesDeleted(Collection<String> arg0) {
   // Called when a roster entries are removed.
 }
@Override
 public void entriesAdded(Collection<String> arg0) {
   // Called when a roster entries are added.
 }
```

```
});
```

[Code 6]

Sending Messages

Messages can be sent in one of two ways:

- In the form of packets using XMPPConnection sendPacket(Message msg) method.
- As a string of Chat Messages using the Chat class. The Chat is a series of messages exchange between two or more users.

Following code snippet shows how to send Message using sendPacket() method of XMPPConnection

```
// Send chat msg to with msg type as (chat, normal, groupchat, headline, error)
Message msg = new Message(String to, Message.Type type);
msg.setBody("How are you?");
connection.sendPacket(msg);
```

[Code 7]

The chat class is a convenient way to send messages. The following code snippet shows how to send message using Chat class.

```
ChatManager chatmanager = connection.getChatManager();
Chat newChat = chatmanager.createChat("abc@gmail.com", new MessageListener() {
  // Receiving Messages
  public void processMessage(Chat chat, Message message) {
    Message outMsg = new Message(message.getBody());
    try {
      //Send Message object
      newChat.sendMessage(outMsg);
    } catch (XMPPException e) {
      //Error
    }
  }
});
try {
  //Send String as Message
  newChat.sendMessage("How are you?");
} catch (XMPPException e) {
  //Error
}
```

[Code 8]

The ChatManager instance is obtained from XMPPConnection using the getChatManager() method. ChatManager keeps track on all current chats. Chat is created which will now be series of messages exchanged between two users.

The sendMessage(String msg) or sendMessage(Message msg) method is used for sending text messages or message object in the context of a given chat session.

Moreover, MessageListener can be used to get callbacks of notification of Message from other users on chat.

Receiving Messages

Receiving messages from other user is done using:

- •Poll mechanism provided through the use of the PacketCollector class.
- •Asynchronous mechanism through the use of the PacketListener (Recommended)

The following code snippet shows an asynchronous way of listening to incoming messages using PacketListener.

```
// Add a packet listener to get messages sent to us
PacketFilter filter = new MessageTypeFilter(Message.Type.chat);
connection.addPacketListener(new PacketListener() {
  public void processPacket(Packet packet) {
    Message message = (Message) packet;
    String body = message.getBody();
    String from = message.getFrom();
  }
}, filter);
```

[Code 9]

Sample Example

Here an example consists of ListView to display the chat conversation between two users. It has by default set values of Google talk XMPP parameters.

Remember to add the jar file into your project lib folder and Internet permission in manifest file. The example is using the asmack library: patched version made for Android.

res\layout\main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >
```

```
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:minWidth="70dp"
      android:text="Chat With"
      android:textStyle="bold" />

    <EditText
      android:id="@+id/toET"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:hint="Add Recipient"
      android:minWidth="250dp"
      android:scrollHorizontally="true"
      android:singleLine="true"
      android:textSize="16sp" />
</LinearLayout>

<ListView
    android:id="@+id/listMessages"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:scrollbars="horizontal" />

<LinearLayout
    android:id="@+id/linearLayout2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="bottom"
    android:orientation="horizontal" >

    <EditText
      android:id="@+id/chatET"
      android:layout_width="0dip"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:gravity="top"
      android:hint="Type to compose"
      android:scrollHorizontally="true" >
    </EditText>

    <Button
```

```
      android:id="@+id/sendBtn"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginLeft="5dp"
      android:layout_marginRight="5dp"
      android:layout_marginTop="5dp"
      android:text="Send"
      android:textStyle="bold" />
  </LinearLayout>
</LinearLayout>
```

[Code 10]

res\layout\listitem.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/text1"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:singleLine="false"
  android:textStyle="bold" />
```

[Code 11]

AndroidManifest.xml

```
"<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.demo.xmppchat"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk android:minSdkVersion="8" />

  <uses-permission android:name="android.permission.INTERNET" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name" >
    <activity
      android:name=".XMPPChatDemoActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
```

```
      </intent-filter>
    </activity>
  </application>

</manifest>
```

[Code 12]

XMPPChatDemoActivity.java

```
package com.demo.xmppchat;

import java.util.ArrayList;
import java.util.Collection;

import org.jivesoftware.smack.ConnectionConfiguration;
import org.jivesoftware.smack.PacketListener;
import org.jivesoftware.smack.Roster;
import org.jivesoftware.smack.RosterEntry;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smack.filter.MessageTypeFilter;
import org.jivesoftware.smack.filter.PacketFilter;
import org.jivesoftware.smack.packet.Message;
import org.jivesoftware.smack.packet.Packet;
import org.jivesoftware.smack.packet.Presence;
import org.jivesoftware.smack.util.StringUtils;

import android.app.Activity;
import android.app.ProgressDialog;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;

public class XMPPChatDemoActivity extends Activity {

  public static final String HOST = "talk.google.com";
  public static final int PORT = 5222;
  public static final String SERVICE = "gmail.com";
  public static final String USERNAME = "userid@gmail.com";
```

```
public static final String PASSWORD = "password";

private XMPPConnection connection;
private ArrayList<String> messages = new ArrayList<String>();
private Handler mHandler = new Handler();

private EditText recipient;
private EditText textMessage;
private ListView listview;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  recipient = (EditText) this.findViewById(R.id.toET);
  textMessage = (EditText) this.findViewById(R.id.chatET);
  listview = (ListView) this.findViewById(R.id.listMessages);
  setListAdapter();

  // Set a listener to send a chat text message
  Button send = (Button) this.findViewById(R.id.sendBtn);
  send.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
      String to = recipient.getText().toString();
      String text = textMessage.getText().toString();
      Log.i("XMPPChatDemoActivity ", "Sending text " + text + " to " + to);
      Message msg = new Message(to, Message.Type.chat);
      msg.setBody(text);
      if (connection != null) {
        connection.sendPacket(msg);
        messages.add(connection.getUser() + ":");
        messages.add(text);
        setListAdapter();
      }
    }
  });
  connect();
}

/**
 * Called by Settings dialog when a connection is establised with
```

```
   * the XMPP server
   */
  public void setConnection(XMPPConnection connection) {
    this.connection = connection;
    if (connection != null) {
      // Add a packet listener to get messages sent to us
      PacketFilter filter = new MessageTypeFilter(Message.Type.chat);
      connection.addPacketListener(new PacketListener() {
        @Override
        public void processPacket(Packet packet) {
          Message message = (Message) packet;
          if (message.getBody() != null) {
            String fromName = StringUtils.parseBareAddress(message.getFrom());
            Log.i("XMPPChatDemoActivity ", " Text Recieved " + message.getBody() + " from
" +  fromName);
            messages.add(fromName + ":");
            messages.add(message.getBody());
            // Add the incoming message to the list view
            mHandler.post(new Runnable() {
              public void run() {
                setListAdapter();
              }
            });
          }
        }
      }, filter);
    }
  }

  private void setListAdapter() {
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.listitem,
messages);
    listviewr.setAdapter(adapter);
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    try {
      connection.disconnect();
    } catch (Exception e) {

    }
```

```
  }

  public void connect() {

    final ProgressDialog dialog = ProgressDialog.show(this, "Connecting...", "Please
wait...", false);
    Thread t = new Thread(new Runnable() {
      @Override
      public void run() {
        // Create a connection
       ConnectionConfiguration connConfig = new ConnectionConfiguration(HOST, PORT,
SERVICE);
       XMPPConnection connection = new XMPPConnection(connConfig);
         try {
           connection.connect();
           Log.i("XMPPChatDemoActivity",  "[SettingsDialog] Connected to
"+connection.getHost());
         } catch (XMPPException ex) {
             Log.e("XMPPChatDemoActivity",  "[SettingsDialog] Failed to connect to "+
connection.getHost());
             Log.e("XMPPChatDemoActivity", ex.toString());
             setConnection(null);
         }
          try {
           connection.login(USERNAME, PASSWORD);
           Log.i("XMPPChatDemoActivity",  "Logged in as" + connection.getUser());

           // Set the status to available
           Presence presence = new Presence(Presence.Type.available);
           connection.sendPacket(presence);
           setConnection(connection);

           Roster roster = connection.getRoster();
           Collection<RosterEntry> entries = roster.getEntries();
           for (RosterEntry entry : entries) {

             Log.d("XMPPChatDemoActivity",  "--------------------------------------");
             Log.d("XMPPChatDemoActivity", "RosterEntry " + entry);
             Log.d("XMPPChatDemoActivity", "User: " + entry.getUser());
             Log.d("XMPPChatDemoActivity", "Name: " + entry.getName());
             Log.d("XMPPChatDemoActivity", "Status: " + entry.getStatus());
             Log.d("XMPPChatDemoActivity", "Type: " + entry.getType());
             Presence entryPresence = roster.getPresence(entry.getUser());
```

```
                Log.d("XMPPChatDemoActivity", "Presence Status: "+
entryPresence.getStatus());
                Log.d("XMPPChatDemoActivity", "Presence Type: " + entryPresence.getType());

                Presence.Type type = entryPresence.getType();
                if (type == Presence.Type.available)
                  Log.d("XMPPChatDemoActivity", "Presence AVIALABLE");
                  Log.d("XMPPChatDemoActivity", "Presence : " + entryPresence);
                }
                } catch (XMPPException ex) {
                  Log.e("XMPPChatDemoActivity", "Failed to log in as "+  USERNAME);
                  Log.e("XMPPChatDemoActivity", ex.toString());
                  setConnection(null);
                }
                dialog.dismiss();
              }
        });
    t.start();
    dialog.show();
  }
}
```

[Code 13]

ref:http://developer.samsung.com/android/technical-docs/Building-a-Chat-Application