Communication-Overview

1. First the message is stored locally on the Android device. In the local SQLite Database. There is a local _id supplied for the listviews, but the BackendID which is supplied by the server is set to 0.

SingleChatActivity.java

```
Message = (EditText) findViewById(R.id.chatText);
        Button send = (Button)
findViewById(R.id.buttonSend);
        send.setOnClickListener(new View.OnClickListener() {
            public void onClick(View arg0) {
                if (Message.getText().toString().length() >
0) {

                    Intent iSendTextMsgService = new
Intent(SingleChatActivity.this, MeBaService.class);


iSendTextMsgService.setAction(Constants.ACTION_SENDTEXTMESSA
GE);

iSendTextMsgService.putExtra(Constants.CHATNAME, ChatName);

iSendTextMsgService.putExtra(Constants.CHATID, ChatID);

iSendTextMsgService.putExtra(Constants.USERID, userid);
                    String tmpmsg =
Message.getText().toString();

iSendTextMsgService.putExtra(Constants.TEXTMESSAGE, tmpmsg);

                    startService(iSendTextMsgService);

                    Message.setText("");
                    Message.clearFocus();
                }
            }
        });
```

MeBaService.java

```
private void insertNewMsgIntoDB(int ChatID, int UserID,
String MessageType, String Message) {
        Log.d(TAG, "start insertMsgIntoDB");

        // Insert new Message into local DB and trigger Sync
to upload the Information.
        // To find the not send messages the Backend ID
musst be 0 and the
        // Sendtimestamp musst be 0
        // The Readtimestamp and the MessageIDs are supplied
by the Server
```

```
        // The ChatID is needed to insert the Message into
the right Chat afterwards

        long time = System.currentTimeMillis() / 1000L;

        ContentValues valuesins = new ContentValues();
        valuesins.put(Constants.T_MESSAGES_BADBID, 0);
        valuesins.put(Constants.T_MESSAGES_OwningUserID,
UserID);
        valuesins.put(Constants.T_MESSAGES_OwningUserName,
username);
        valuesins.put(Constants.T_MESSAGES_ChatID, ChatID);
        valuesins.put(Constants.T_MESSAGES_MessageTyp,
MessageType);
        valuesins.put(Constants.T_MESSAGES_SendTimestamp,
time);
        valuesins.put(Constants.T_MESSAGES_ReadTimestamp,
time);
        if
(MessageType.equalsIgnoreCase(Constants.TYP_TEXT)) {
            valuesins.put(Constants.T_MESSAGES_TextMsgID,
0);
            valuesins.put(Constants.T_MESSAGES_TextMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_IMAGE)) {
            valuesins.put(Constants.T_MESSAGES_ImageMsgID,
0);

valuesins.put(Constants.T_MESSAGES_ImageMsgValue, Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_LOCATION)) {

valuesins.put(Constants.T_MESSAGES_LocationMsgID, 0);

valuesins.put(Constants.T_MESSAGES_LocationMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_CONTACT)) {
            valuesins.put(Constants.T_MESSAGES_ContactMsgID,
0);

valuesins.put(Constants.T_MESSAGES_ContactMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_FILE)) {
            valuesins.put(Constants.T_MESSAGES_FileMsgID,
0);
            valuesins.put(Constants.T_MESSAGES_FileMsgValue,
Message);
        } else if
(MessageType.equalsIgnoreCase(Constants.TYP_VIDEO)) {
```

```
            valuesins.put(Constants.T_MESSAGES_VideoMsgID, 
0);

valuesins.put(Constants.T_MESSAGES_VideoMsgValue, Message);
        }
        ContentProviderClient client = 
getContentResolver().acquireContentProviderClient(FrinmeanCo
ntentProvider.MESSAES_CONTENT_URI);

client.getLocalContentProvider().insert(FrinmeanContentProvi
der.MESSAES_CONTENT_URI, valuesins);

        Log.d(TAG, "end insertMsgIntoDB");
    }
```

2. the 0 in the BackendID is a trigger for the SyncService, these message is unknown to the server and need to be sent to the server (uploadUnsavedMessages).

SyncAdapter.java

```
private void uploadUnsavedMessages() {

        ContentProviderClient client = 
mContentResolver.acquireContentProviderClient(FrinmeanConten
tProvider.MESSAES_CONTENT_URI);
        Cursor c = ((FrinmeanContentProvider) 
client.getLocalContentProvider()).query(FrinmeanContentProvi
der.MESSAES_CONTENT_URI, MESSAGES_DB_Columns, 
T_MESSAGES_BADBID + " = ?", new String[]{"0"}, null);

        while (c.moveToNext()) {

            String msgtype = 
c.getString(ID_MESSAGES_MessageType);

            if (msgtype.equalsIgnoreCase(TYP_TEXT)) {
                OutSendTextMessage outtxt = 
rf.sendtextmessage(username, password, 
c.getString(ID_MESSAGES_TextMsgValue));
                if (outtxt != null) {
                    if (outtxt.getErrortext() == null || 
outtxt.getErrortext().isEmpty()) {
                        OutInsertMessageIntoChat outins = 
rf.insertmessageintochat(username, password, 
c.getInt(ID_MESSAGES_ChatID), outtxt.getTextID(), TYP_TEXT);
                        if (outins != null) {
                            if (outins.getErrortext() == 
null || outins.getErrortext().isEmpty()) {

updateUploadedNessagesDatabase(c.getInt(ID_MESSAGES__id), 
outins.getMessageID(), outins.getSendTimestamp(), 
outins.getSendTimestamp(), outtxt.getTextID(), TYP_TEXT, 
null);
                            }
                        }
                    }
                }
            }
```
…….

Der Server legt pro User im Chat einen eigenen Eintrag ind er Messages Tabelle an. Deswegen 2 Nachrichten auf dem Server, eine für den Versender, eine für den Empänger.
Der Content wird vor dem Einfügen der Nachricht in den Chat hoch geladen, z.B. für Text mit SendTextMessage und dem Return-Type OutSendTextMessage. (Trennung Content und Chat-Inhalt).

3.

The SyncAdapter is perodically checking for new messages with (syncCheckNewMessages). If there are new messages in any chat, the message inclusive the content will be loaded from the server.
The funktione SaveMessageToDB takes care that all messages are loaded in a loop and for each message the content is loaded, see:

```
private void SaveMessageToLDB(List<Message> in, int ChatID, 
String ChatName) {
        Log.d(TAG, "start SaveMessageToLDB");
        for (int j = 0; j < in.size(); j++) {
            Message m = in.get(j);
            ContentValues valuesins = new ContentValues();
            valuesins.put(T_MESSAGES_BADBID, 
m.getMessageID());
            valuesins.put(T_MESSAGES_OwningUserID, 
m.getOwningUser().getOwningUserID());
            valuesins.put(T_MESSAGES_OwningUserName, 
m.getOwningUser().getOwningUserName());
            valuesins.put(T_MESSAGES_ChatID, ChatID);
            valuesins.put(T_MESSAGES_MessageTyp, 
m.getMessageTyp());
            valuesins.put(T_MESSAGES_SendTimestamp, 
m.getSendTimestamp());
            valuesins.put(T_MESSAGES_ReadTimestamp, 
m.getReadTimestamp());
            if 
(m.getMessageTyp().equalsIgnoreCase(TYP_TEXT)) {
                valuesins.put(T_MESSAGES_TextMsgID, 
m.getTextMsgID());
                OutFetchTextMessage oftm = 
rf.gettextmessage(username, password, m.getTextMsgID());
                if (oftm != null) {
                    if (oftm.getErrortext() == null || 
oftm.getErrortext().isEmpty()) {
                        if 
(acknowledgeMessage(Constants.TYP_TEXT, 
oftm.getTextMessage(), m.getMessageID())) {

valuesins.put(T_MESSAGES_TextMsgValue, 
oftm.getTextMessage());
```

```
                            ContentProviderClient client = 
mContentResolver.acquireContentProviderClient(FrinmeanConten
tProvider.MESSAES_CONTENT_URI);
                            ((FrinmeanContentProvider) 
client.getLocalContentProvider()).insertorupdate(FrinmeanCon
tentProvider.MESSAES_CONTENT_URI, valuesins);
                            client.release();
                        }
                    }
                }
            }
…….
```

4.

The Acknowledge of a message is setting the ReadTimestamp on the server, bevor only the TempReadTimestamp was set when the message was delivered to the client. If there is a probleme with the Acknowledge, the ReadTimestamp will stay at 0 and at every time the server is asked for new messages, the message will be supplied again to the client.

```
……..
if (m.getMessageTyp().equalsIgnoreCase(TYP_TEXT)) {
                valuesins.put(T_MESSAGES_TextMsgID, 
m.getTextMsgID());
                OutFetchTextMessage oftm = 
rf.gettextmessage(username, password, m.getTextMsgID());
                if (oftm != null) {
                    if (oftm.getErrortext() == null || 
oftm.getErrortext().isEmpty()) {
                        if 
(acknowledgeMessage(Constants.TYP_TEXT, 
oftm.getTextMessage(), m.getMessageID())) {

valuesins.put(T_MESSAGES_TextMsgValue, 
oftm.getTextMessage());
……
```

Hier ein Teil der Acknowledge Funktion mit dem Aufruf des Restful Service

```
private boolean acknowledgeMessage(String msgType, String 
message, int msgid) {
        Log.d(TAG, "start acknowledgeMessage");

        boolean ret = false;

        if (msgType.equalsIgnoreCase(Constants.TYP_TEXT)) {
            Hasher hasher = Hashing.sha1().newHasher();
            hasher.putBytes(message.getBytes());
            byte[] sha1 = hasher.hash().asBytes();

            OutAcknowledgeMessageDownload oack = 
rf.acknowledgemessagedownload(username, password, msgid, 
sha1.toString());
            if (oack != null) {
                if (oack.getErrortext() == null || 
oack.getErrortext().isEmpty()) {
```

```
                    if
(oack.getAcknowledge().equalsIgnoreCase(Constants.ACKNOWLEDG
E_TRUE)) {
                        ret = true;
                    }
                }
            }
        }
```
……….

5.

To get the status of a message, all rows in the db needs to be taken in account. At the sending of a message 2 – n rows are created. The message of the sender has automatically a Send-, Read- amd ShowTimestamp. The SendTimestamp is set in the Action shown above with the number 2. The ReadTimestamp is set in the Action shown above with the number 4. If a user is opening a chat on the frontend, all messages where the ShowTimestamp is 0 are collected and the ShowTimestamp is then set. Now we have a base for the status of the message by collecting the counts of Readtimestamp and showtimestamps for messages with the same OriginMsgID (= all messages are created by one send)