Movino --> Next2Friends (java)

*Author: Lawrence Botley  
Date: 27th July 2007  
Version: 001*

**Abstract**

This document outlines the high level changes that need to be made to the Java version of Movino in order to become the Next2Friends software for J2ME devices.

**Index**

1. Global class changes
2. Change logos, text and assemble ownership labelling and information
3. Implement Bluetooth matching protocol over TCP/IP based Personal Area Network
4. Encrypt messages using the MD5 algorithm
5. Send match Messages to the central server
6. Alter MediaPacket to comply with Red5 RTMP data packet
7. Encrypt/descript and store user data in local XML file
8. Listen and authenticate and reply to incoming server requests for media
9. Generate progressive JPEG thumbnails of photos and video frames

# 1. Global Class changes

1. MovinoPacket should be renamed MediaPacket
2. An additional Class named FriendPacket should be created and hold the data structure of

* Local users 128 bit unique identifier
* The date and time of now
* An encrypted message containing the connecting users 128 bit unique identifier and date encrypted by the local users private encryption key. (note: this data item will be empty if the local user is the connection initialise as they will have no knowledge of the remote users 128 bit unique identifier)

The FriendPacket will be exchanged between devices over Bluetooth

1. ConnectionThread will be split into two classes FriendConnectionThread and ServerConnectionThread.

* FriendConnectionThread.java – threads for bluetooth communications
* ServerConnectionThread.java – threads for central server communications

# 2. Change logos, text and assembly ownership labelling and information

**Overview**

Namespaces, variables, classes, splash screen ll contain *Movino* branding. This should all be completely removed and replaced by *Next2Friends* branding. The rule of thumb is; if the class source or file name contains the word ***Movino****,* then change it to***Next2Friends***

Java Class files: **SplashScreen.java**

* Line 42: Logo to be changed to our own
* All references to Movino Should be changed to Next2Friends
* Where ever else found in the application

# 3. Implement Bluetooth matching protocol over TCP/IP based Personal Area Network

**Overview**

A search must begin as soon as the application is fired up (user login details permitting).

Java Class files: **BluetoothSearch.java**

* Line 123: Upon successfully finding a device, it shall be uniquely identified and a lookup in the XML document will determine if there has been at least 1 minute since the last match. If it hasn’t been a match in the last 1 minute then an attempt will be made to exchange profiles with the user. This will be made by calling xx

# 4. Encrypt messages using the MD5 algorithm Network

**Overview**

A message containing the remote users 128 bit unique identifier along with the date and time. This will be encrypted using the users private encryption key that is stored in the XML file. These details are encrypted using a private and unique key that is store in the assemble itself. This key will be generated at initial runtime and can be updated remotely by the central server.

Note: For early development stages, if embedding the key in the assembly proves difficult, an interim solution would be to store this in the XML file.

Java Class files: **MD5.java**

* The MD5.java class file provides encryption routines and can return an array of encrypted bytes

# 5. Send match Messages to the central server

**Overview**

In order to synchronisze the users match with the central server, all the FriendPackets much be uploaded. Firstly, the application must send username and password details to the server, once authenticated the application will stream the FriendPackets until. After which they are flagged as sent in the XML document and not uploaded next time.

Java Class files: **ServerConnectionThread.java, HandShakePacket.java**

* The ServerConnectionThread.java will control all communication and authentication with the server including:
  + Uploading new matches in binary
  + Sending photos and videos in binary
  + Streaming Live video and audio to the server

HandShakePacket.java provides a setup with the server and authentication. If possible the MD5 encryotion should be temorarily removed for development

Note: ServerConnectionThread.java already allows the streaming of Live Video to the server. The protocol must to altered to addionally take an audio stream , uploading the matches and media files as well as provide autentication

# 6. Alter MediaPacket to comply with Red5 RTMP data packets

For initial stages of development the MediaPacket will remain in their current state

# 7. Encrypt/decrypt and store user data in local XML file

**Overview**

Settings.xml is a new config/status file that holds the users details, application state and match details.

Java Class files: **XMLConfig.java**

* The XMLConfig.java handle all reads and writes to the XML document stored on disk in the same dirctory as the executable assembly.
* The MD5.java class file provides encryption routines that should encrypt and decrypt the following:
  + UserName
  + Password
  + Unique Identifier
  + Key <- this key is randomly created at run time and stored securely in the assembly

The following is example of this XML document and the application should be able to read and write these details.

<?xml version="1.0" encoding="utf-8"?>

<N2F ConfigVersion="0.1"> // the config file version number

<Username>Lawrence5784</Username> // login nickname

<Password>Secret123</Password> // login password

<HistoryKey>4C4E80F6-BD32-4e96-882A-3153DD211E0E </ HistoryKey > // The mutually remembered key from the last connection

<Identifier>555E4661-93A1-474f-80EE-48C1E27ED2B4</Identifier > // 128 bit unique identifier

<Key>B8917530-C68D-41cc-AE7A-462CBB57B909</Key> // unique encryption key for friend matching

<Options>

<Option1>0</Option1> // miscellaneous options

<Option2>1</Option2>

<Option3>0</Option3>

<Option4>0</Option4>

</Options>

<Matches>

<Match>

<MatchID>9B56CE28-09A1-4762-A4B4-C0E03CC2988B</MatchID> // 128 bit unique identifier of the match

<Certificate ConfirmedByServer="False">EFRE7V74BVKU54UV7VB7VBU76BU76U6732NF5356V3F06</Certificate >

<Certificate ConfirmedByServer="False">3875DYM85C70YXC57YD248XCY805424Y5C7N48F0XC7YN</Certificate >

<Certificate ConfirmedByServer="False">R342UM5YCNT74Y7MXY547XY74YX74YFX80MY70X29T895</Certificate >

<Certificate ConfirmedByServer="True">CX3F7H7F30XMF3Y820XM0EWQ84FMXC3478Q2034XCN8470</Certificate >

<Certificate ConfirmedByServer="True">5467H2CN5F74H80237XC42HF0X8QW7FN023HFM-XC8F73</Certificate >

// serveral matches have been made with this user. The Acii text is the ecrytped certificate

// The Certificates are flagged ConfirmedToServer try after the server confirms with a receipt

</Match>

<Match>

<MatchID>4C4E80F6-BD32-4e96-882A-3153DD211E0E</MatchID>

<Certificate ConfirmedByServer="False">EFRE7V74BVKU54UV7VB7VBU76BU76U6732NF5356V3F06</Certificate >

<Certificate ConfirmedByServer="True">3875DYM85C70YXC57YD248XCY805424Y5C7N48F0XC7YN</Certificate >

<Certificate ConfirmedByServer="True">R342UM5YCNT74Y7MXY547XY74YX74YFX80MY70X29T895</Certificate >

<Certificate ConfirmedByServer="True">GVTRGV63CV56H675VV3578KVC426Y24UY76OK986KO890O</Certificate >

<Certificate ConfirmedByServer="True">5Y43V489TRERCFRE4N9HCF4Y893475CFN54G7C4GNF8540C</Certificate >

<Certificate ConfirmedByServer="True">CX3F7H7F30XMF3Y820XM0EWQ84FMXC3478Q2034XCN8470</Certificate >

<Certificate ConfirmedByServer="True">5467H2CN5F74H80237XC42HF0X8QW7FN023HFM-XC8F73</Certificate >

</Match>

<Match>

<MatchID>E0065B0E-31DA-4e0b-B2B0-36E52983FF8D</MatchID>

<ConfirmedByServer>false</ConfirmedByServer>

<Certificate ConfirmedByServer="False">EFRE7V74BVKU54UV7VB7VBU76BU76U6732NF5356V3F06</Certificate >

<Certificate ConfirmedByServer="False">3875DYM85C70YXC57YD248XCY805424Y5C7N48F0XC7YN</Certificate >

<Certificate ConfirmedByServer="True">CX3F7H7F30XMF3Y820XM0EWQ84FMXC3478Q2034XCN8470</Certificate >

<Certificate ConfirmedByServer="True">5467H2CN5F74H80237XC42HF0X8QW7FN023HFM-XC8F73</Certificate >

</Match>

</Matches>

</N2F>

# 8. Listen and authenticate and reply to incoming server requests for media

**Overview**

Users may browse thumbnails of media files on their devices online. To do this, the central server will make a request to the device application with a request to list all file names and types. The sever may then request each individual file to have a thumbnail created and transmitted back to the server where it is then pass on to the users browser. From here users may request the original file, this can also be transmitted to the server.

Only a specific IP range will be allowed to communicate with the listening device, all others attempts should be ignored. The central server will provide a unique 128 bit history key upon teardown of the connection. This will be used next time the server makes a request and provides authenticity. The application will be assigned an initial key when the user first logs in. From here on, new keys will be made on every connection and remembered both ends as a mutual passkey for communication

Java Class files: **ServerConnectionThread.java**

* **ServerConnectionThread.java** is responsible for handling uloading of data to the server as well as listening out for commands to upload media.

9. Generate progressive JPEG thumbnails of photos and video  
  
**Overview**

When the server requests a thumbnail from the device it must find the original and simple scale it so to around 150px width (although this must be easily configurable)

The JDK framework can be used for this. Simple photos much be encoded into JPEGs and since the movie is stored as JPEG frames also, the 5th frame will be taken and be transformed into a scaled progressive JPEG thumbails of also 150px width.

Java Class files: **Thumbnail.java**

* **Thumbnail.java** provides byte streams of scaled photos and video frames from disk