Kode Sumber Hub

Socket.js

```
var net = require("net");
var sqlite3 = require("sqlite3");
var db = new
sqlite3.Database('./trackerdb',
sqlite3.OPEN_READWRITE);

//untuk simpan socket yang terbuka (per
client)
var client = [];
function start() {
    var server = net.createServer(function
(socket) {
        socket.setEncoding("UTF8")
        socket.name = socket.remoteAddress;
        console.log("Connection from : " +
socket.remoteAddress);
        client.push(socket); // simpan
socket ke array client
        var clName = socket.name;
        socket.on("error", function (error)
{
            console.log(error);
        });
        socket.on("data", function (data) { {
            var command = ['FL', 'FD', 'NA',
'UP', 'UN'];
         data = data.replace(/
(\r\n|\n|\r)/gm, ""); //get rid line break
at last char
            var req =
data.toString().split(";;");
            //handle command request
            if (req[0] == "FL") {
                fileList();
            } else if (req[0] == "FD")
                fileDetail(req[1]);
            else if (req[0] == "NA")

nodeActive(socket.remoteAddress);
            else if (req[0] == "UP") {
                // get id_host from ip
                db.each("SELECT id_host FROM
host WHERE ip = '" + socket.remoteAddress
+ "'",
                    function (err, row) {
                        var id_host =
row.id_host;

                        req.push(id_host);
                        nodeUpdate(req);
                    } ); }
            else if (req[0] == "UN") {
                console.log("Request
isinya : "+req);
                db.each("SELECT id_host FROM
host WHERE ip = '" + socket.remoteAddress
+ "'",
                    function (err, row) {
                        var id_host =
row.id_host;

                        req.push(id_host);
                        db.each("SELECT
id_file FROM file WHERE nama = '" + req[1]
+ "'",function (err, row) {
                            var id_file =
row.id_file;

                            req.push(id_file);

                            nodeUpdateName(req);

console.log("Request isinya : "+req);
                        });} );}
            else if(req[0] == "ADD")
                add(req);
            else
                socket.write("Unspecified
Command");
            //command request function
            function fileList() {
                db.serialize(function () {
                    db.all("SELECT
f.id_file,f.nama,f.bitrate,f.samplerate,f.
size FROM file as f", function (err, row)

socket.write(JSON.stringify(row));
                    }); }); }
            //get file detail from active
host

            function fileDetail(fileId) {
                db.serialize(function () {
                    var query = "
SELECT file.nama , host.ip,
file_host_rel.block_avail FROM
file_host_rel \
INNER JOIN file ON file_host_rel.id_file =
file.id_file \
INNER JOIN host ON file_host_rel.id_host =
host.id_host \
WHERE host.active = 1 AND file.id_file = "
+ fileId;
                    db.all(query, function
(err, row) {

socket.write(JSON.stringify(row));
                    });});}
            // update node active status
             function nodeActive(nodeId) {
                db.serialize(function () {
                    db.all("SELECT * FROM
host WHERE ip='" + nodeId + "'", function
(err, row) { var time = new Date;
                        if (row.length < 1) {
                            // add host if we
can't find it
                            db.run("INSERT
INTO host (ip,active,time) VALUES ('" +
```

```
nodeId + "',1," + time.getTime() + ")");
                    } else
                    // set node status
to active
                        db.run("UPDATE
host SET active= 1, time= " +
time.getTime() + " WHERE ip='" + nodeId +
"'");
                    console.log(nodeId +
" active");
                })
            });
            socket.write("Node " +
nodeId + " is nodeActive");
        }
        // info array of data : 1
id_file, 2 avail_block
        function nodeUpdate(info) {
            db.serialize(function () {
                console.log(info[3] + "
" + socket.remoteAddress);
                db.all("SELECT id_file,
id_host FROM file_host_rel where id_file =
" + info[1] + " AND id_host = " + info[3],
                    function (err, row) {
                        if (row ==
undefined || row.length < 1) {
                            var query =
"INSERT INTO file_host_rel(id_file,
id_host,block_avail) VALUES (" + info[1] +
"," + info[3] + "," + info[2] + ")"

console.log(query);
                            db.all(query,
                                function
(err, row) {
                                    if
(err)

console.log('');});} else {
                            var query =
"UPDATE file_host_rel SET block_avail = "
+ info[2] + " WHERE id_file = " + info[1]
+ " AND id_host= " + info[3];

console.log(query);
                            db.all(query,
                                function
(err, row) {
                                    if
(err)

console.log('');} );}});});}
        function nodeUpdateName(info) {
        // 0 UN | 1 Filename | 2
block_avail | 3 id_host | 4 id_file
        for (var i = 0; i < info.length;
i++) {
            console.log(" UNInfo[" + i +
"] : " + info[i]);
        }
            db.serialize(function () {
                console.log(info[3] + "
" + socket.remoteAddress);
                db.all("SELECT id_file,
id_host FROM file_host_rel where id_file =
" + info[4] + " AND id_host = " + info[3],
                    function (err, row) {
                        if (row ==
undefined || row.length < 1) {
                            var query =
"INSERT INTO file_host_rel(id_file,
id_host,block_avail) VALUES (" + info[4] +
"," + info[3] + "," + info[2] + ")"

console.log(query);
                            db.all(query,
                                function
(err, row) {
                                    if
(err)

console.log(''); });
                        } else {
                            var query =
"UPDATE file_host_rel SET block_avail = "
+ info[2] + " WHERE id_file = " + info[4]
+ " AND id_host= " + info[3];

console.log(query);
                            db.all(query,
                                function
(err, row) {
                                    if
(err)

console.log('');
                                });}});});
    }
    // add new file
    function add(info) {
        //filename, bitrate,
samplerate, size
        var query = "INSERT INTO
file(nama,bitrate,samplerate,size)
VALUES('"+info[1]+"','+info[2]+','+info[3]
+','+info[4]+')";
        db.serialize(function () {
            db.run(query);
        });
        console.log("ADD "+query);
    }});});
// set timeout 5 minutes (30000)
// run node active checking
function isActive() {
    var now = new Date;
    db.serialize(function () {
        console.log("run is active");
        db.each("select * from host",
function (err, row) {
```

```
            var diff = now.getTime() -
row.time;
            //console.log(diff+"
"+now.getTime()+" "+row.time);
            if (diff > 600000) { // more
than 10 minutes
                db.run("UPDATE host SET
active = 0 WHERE ip = '" + row.ip + "'");
                console.log(row.ip + "
is inactive\n");
            } else
                console.log(row.ip + "
is active\n");
        });});}
    setInterval(isActive, 300000);
    server.listen(1337);
}
exports.start = start
```

Kode Sumber NodePeer

AsynchRedServ.cs

```
using System;using
System.Collections.Generic;using
System.Net.Sockets;using System.Net;using
System.Threading;using System.Text;using
System.IO;
namespace upikapik
{
    class AsynchRedServ : IDisposable
    {
        private string FILE_DIR =
AppDomain.CurrentDomain.BaseDirectory +
"music/";
        private const int MSG_LENGTH_BYTE =
100; //masih ngasal, panjang dari request
message
        private int threadCount = 0;
        private TcpListener server;
        public bool enable = true;
        private IPEndPoint ipServer;
        private ManualResetEvent allDone =
new ManualResetEvent(false); // to track
thread state

        public AsynchRedServ()
        {
            ipServer = new
IPEndPoint(IPAddress.Parse("127.0.0.1"),
1338);
        }
        public AsynchRedServ(string
ipAddress, int port)
        {
            ipServer = new
IPEndPoint(IPAddress.Parse(ipAddress),
port);
        }

        public void startListening()
        {
            try
            {
                server = new
TcpListener(ipServer);
                server.Server.ReceiveTimeout
= 8000;
                server.Start();
                while (enable){
                    allDone.Reset();

server.BeginAcceptTcpClient(acceptCallback
, server);
                    allDone.WaitOne();
                }
            }
            catch (Exception e)
            {

Console.WriteLine(e.ToString());
            }
        }
        private void
acceptCallback(IAsyncResult result)
        {
            allDone.Set();
            string command;
            string[] parsedCommand;
            string filename;
            byte[] buffRead = new
byte[MSG_LENGTH_BYTE];
            byte[] buffSend;
            int startPost;
            int size;
            TcpListener server =
(TcpListener)result.AsyncState;
            TcpClient client = null;
            try
            {
                client =
server.EndAcceptTcpClient(result);
            }
            catch (ObjectDisposedException
ex)
            {
                return;
            }
            NetworkStream clientStream =
client.GetStream();
            if (threadCount >= 5)
            {
                byte[] busy =
System.Text.Encoding.UTF8.GetBytes("BUSY")
;
                clientStream.Write(busy, 0,
busy.Length);
            } else
            {
```

```
            try
            {
clientStream.Read(buffRead, 0,
MSG_LENGTH_BYTE);
                command =
System.Text.Encoding.UTF8.GetString(buffRe
ad);

//GET;filename;block_start,size
                parsedCommand =
command.Split(';');

                filename =
parsedCommand[1];
                startPost =
Convert.ToInt32(parsedCommand[2]);
                size =
Convert.ToInt16(parsedCommand[3]);
                buffSend = new
byte[size];

                buffSend =
getblocks(filename, startPost, size);
                //send

clientStream.Write(buffSend, 0, size);
            } catch (TimeoutException
ex)
            {
                clientStream.Close();
                client.Close();
            }    }
        clientStream.Close();
        client.Close();
    }
    // get blocks from files
    private byte[] getblocks(string
filename, int startPost, int size)
    {
        byte[] blocks;
        blocks = new byte[size];
        FileStream file = null;
        try
        {
            file = new
FileStream(FILE_DIR + filename,
FileMode.Open, FileAccess.Read,
FileShare.ReadWrite);
            file.Seek(startPost, 0);
            file.Read(blocks, 0, size);
            file.Close();
        }
        catch (FileLoadException ex)
        {

Console.WriteLine(ex.ToString());
        }
        finally
        {
((IDisposable)file).Dispose();
        }

        return blocks;
    }
    public void Dispose()
    {
        enable = false;
        this.server.Stop();
        allDone.Set();
    }    } }
```

AsynchRedStream.cs

```
using System;using System.Collections;using
System.Collections.Generic;using
System.Net.Sockets;using System.Net;using
System.Threading;using System.Text;using
System.IO;using
System.Runtime.InteropServices;
namespace upikapik
{
    // declare it and use startstream and
then stopstream
    class AsynchRedStream
    {
        IPAddress own =
IPAddress.Parse("192.168.0.7");
        file_list fileinfo;
        private const int MAX_REQUEST = 10;
int id_file; byte[] bassBuffer;

        private ManualResetEvent allDone =
new ManualResetEvent(false);
        private object createReqLocker =
new object();
        private object writeBufferLocker =
new object();
        private object writeFileLocker =
new object();
        private bool enable = false;
private bool enStream = false;
        private int startpost = 0;

        private Timer startTimer;
        private Queue<RequestProp>
writeQueue = new
Queue<RequestProp>(MAX_REQUEST);
        private SortedList bufferList = new
SortedList();
        private SortedList writeList = new
SortedList();
        private Queue<RequestProp>
bassBufferQueue = new
Queue<RequestProp>();
        private Queue<RequestProp>
failedRequestQueue = new
Queue<RequestProp>();
```

```
        private Queue<int> starpostQueue =
new Queue<int>();
        private Queue<Hosts> hosts;
        private SortedList writedStartpost
= new SortedList();

        private static int lastAdjacentKey
= 0;
        private static int bufferTurn = 0;
        private static int writeTurn = 0;
        int blocksize;

        FileStream file = null;
        ManualResetEvent manualEvent = new
ManualResetEvent(false);

        // variable to hold test value
        static int ignoredConnection;
        List<string> requestAndHost = new
List<string>();

        public AsynchRedStream()
        {
        }
        public void startStream(int
id_file, file_list fileinfo)
        {
            this.fileinfo = fileinfo;
            this.id_file = id_file;
            string filename =
getFilename();
            blocksize = getBlockSize();
            int filesize = getFileSize();
            bassBuffer = new
byte[filesize];

            file = new
FileStream(AppDomain.CurrentDomain.BaseDir
ectory + "music/" + filename,
FileMode.OpenOrCreate,
FileAccess.Write,FileShare.ReadWrite);

            enable = true;
            enStream = true;
            createStartPostQueue(blocksize,
filesize);
            startTimer = new Timer(x => {
startTimerCallback(filename, blocksize,
filesize); }, null, 0, 500); // is it
better than forever while?
        }
        private void
startTimerCallback(string filename, int
blocksize, int filesize)
        {
            // keep timer always alive
            startTimer.Change(0, 500);
            if (enable){
                if(writeList.Count !=
MAX_REQUEST) {
                    if (enStream) {
                        if
(failedRequestQueue.Count != 0)

startConnect(failedRequestQueue.Dequeue())
;
                        else if
(starpostQueue.Count != 0)

startConnect(createRequest(filename,
blocksize, filesize));
                    }
                    lock (writeBufferLocker)
                    {
                        if (writeList.Count !
= 0)
                            writeToFile();
                        else if
(writeList.Count == 0 && enStream ==
false) {
                            stopStream();

startTimer.Dispose();
                        }
                        writeToBuffer();
                    }}}
                else
                    reset();
            }
            private void
startConnect(RequestProp req)
            {
                req.client = new TcpClient();

                allDone.Reset();
                try
                {

req.client.BeginConnect(req.peer.Address,
req.peer.Port, connectCallback, req);  }
                catch (SocketException ex)
                {
                    enqueueFailedRequest(req);
}

                allDone.WaitOne();
            }
            private void
connectCallback(IAsyncResult result) //
connect to another peer and send message
to get block
            {
                allDone.Set();
                RequestProp req =
(RequestProp)result.AsyncState;
                req.client.EndConnect(result);
                req.stream =
req.client.GetStream();

                byte[] sendBuffer =
Encoding.UTF8.GetBytes("GET;" +
```

```
req.filename + ";" + req.startPost + ";" + 0)
req.blockSize + ";");
            req.receiveBuffer = new
byte[req.blockSize];
            try
            {
                req.stream.Write(sendBuffer,
0, sendBuffer.Length);
            } catch (IOException ex)
            {
                enqueueFailedRequest(req);
            }

            try
            {

req.stream.BeginRead(req.receiveBuffer, 0,
req.blockSize, readCallback, req);
            }  catch (IOException ex)
            {
                enqueueFailedRequest(req);
            }}
        private void
readCallback(IAsyncResult result) // read
response from another peer and write it to
file
        {
            RequestProp req =
(RequestProp)result.AsyncState;
            req.stream.EndRead(result);
            byte[] receiveBuffer = new
byte[req.blockSize];
            receiveBuffer =
req.receiveBuffer;
            lock (writeBufferLocker)
            {
                //writeQueue.Enqueue(req);
                writeList.Add(req.startPost,
req);

bufferList.Add(req.startPost, req);
                if(enable == false)
                {
                    reset();
                }}}
        private void writeToFile() // write
to file
        {
            RequestProp req =
(RequestProp)writeList.GetByIndex(0);

requestAndHost.Add(req.peer.Address.ToStri
ng());
            if (req.startPost == writeTurn)
            {
                try
                {
                    lock (writeFileLocker)
                    {
                        if (req.startPost ==
0)

file.Seek(req.startPost,
SeekOrigin.Begin);
                        else

file.Seek(req.startPost+1,
SeekOrigin.Begin);

file.Write(req.receiveBuffer, 0,
req.receiveBuffer.Length);

writedStartpost.Add(req.startPost,
req.startPost);

writeList.Remove(req.startPost);
                        writeTurn = writeTurn
+ blocksize + 1;
                    }
                } catch (IOException ex)
                {
                }}}
        private void writeToBuffer()
        {
            if (bufferList.Count != 0) {
                RequestProp tmp =
(RequestProp)bufferList.GetByIndex(0);
                if (tmp.startPost ==
bufferTurn) {
                    int y = 0;
                    for (int i =
tmp.startPost; i < tmp.blockSize - 1 +
tmp.startPost; i++) {
                        if (i ==
bassBuffer.Length)
                            break;
                        bassBuffer[i] =
tmp.receiveBuffer[y];
                        y++;
                    }

bufferList.Remove(tmp.startPost);
                    bufferTurn = bufferTurn
+ blocksize + 1;
                }}}

        // tools methods
        private void printBlocks(byte[]
blocks)
        {

Console.WriteLine(System.Text.Encoding.UTF
8.GetString(blocks));
        }
        private void
createStartPostQueue(int blocksize, int
filesize)
        {
            int startpost = 0;
```

```
starpostQueue.Enqueue(startpost);
            while (true) {
                if ((startpost + blocksize)
> filesize)
                    blocksize = filesize -
startpost + 1;
                else
                    startpost = startpost +
blocksize + 1;

starpostQueue.Enqueue(startpost);
                if (startpost + blocksize >=
filesize)
                    break;
            }
        }
        private RequestProp
createRequest(string filename, int
blocksize, int filesize)
        {
            RequestProp req = new
RequestProp();
            req.blockSize = blocksize;
            req.filename = filename;
            req.startPost =
starpostQueue.Dequeue();
            req.peer = new
IPEndPoint(IPAddress.Any, 1338);

            if ((startpost + blocksize) >
filesize)
                req.blockSize = filesize -
req.startPost + 1;

            lock (createReqLocker)
            {
                // get the address then
enqueue it again
                // if block available from
host smaller than requested, get another
host
                // if the address is same as
our address, continue
                while (true){
                    Hosts peer =
hosts.Dequeue();
                    hosts.Enqueue(peer);
                    if (own.ToString() ==
peer.peer.Address.ToString() )
                        continue;
                    else
                    {
                        req.peer = peer.peer;
                        if ((peer.blockAvail
* blocksize) >= startpost + req.blockSize)
                            break;
                    }}}
            return req;
        }
        private string getFilename()
        {
            return fileinfo.nama;
        }
        private int getFileSize()
        {
            return fileinfo.size;
        }
        private void
enqueueFailedRequest(RequestProp req)
        {
            ignoredConnection++;
            Hosts host = hosts.Dequeue();
            req.peer = host.peer;
            hosts.Enqueue(host);

failedRequestQueue.Enqueue(req);
        }
        private void reset()
        {
            bufferTurn = 0;
            writeTurn = 0;
            writedStartpost.Clear();
            writeList.Clear();
            bufferList.Clear();
        }
        // public method are intended to
invoked by external event
        public void closeFile()
        {
            if (file != null)
                file.Close();
        }
        public int getBlockSize()
        {
            int frameSize = (144 *
fileinfo.bitrate * 1000) /
fileinfo.samplerate;
            return (12000 / frameSize) *
frameSize;
        }
        public byte[] getBuffer()
        {
            return bassBuffer;
        }
        public int getFrameSize()
        {
            int frameSize = (144 *
fileinfo.bitrate * 1000) /
fileinfo.samplerate;
            return frameSize;
        }
        public void
getHostsAvail(Queue<Hosts> hosts)
        {
            this.hosts = hosts;
        }
        public int getLastAdjacentValue()
        {
            int next;
```

```
            int current;
            int blockSize = getBlockSize();
            int last = 0;
            for (int i = lastAdjacentKey; i
< writedStartpost.Count - 1; i++){
                current =
(int)writedStartpost.GetByIndex(i) + 1;
                next =
(int)writedStartpost.GetByIndex(i + 1);
                if ((current + blockSize) ==
next)
                    last = next;
                else
                    lastAdjacentKey = i - 1;
            }
            return last;
        }
        public int
getLastAdjacentBufferValue()
        {
            RequestProp next;
            RequestProp current;
            int blockSize = getBlockSize();
            int last = 0;
            for (int i = 0; i <
bufferList.Count - 1; i++) {
                current =
(RequestProp)bufferList.GetByIndex(i);
                next =
(RequestProp)bufferList.GetByIndex(i + 1);
                if ((current.startPost + 1 +
blockSize) == next.startPost)
                    last = next.startPost;
            }
            return last;
        }
        public int getLastValueOfBuffer()
        {
         if(bufferList.Count != 0){
          RequestProp tmp =
(RequestProp)bufferList.GetByIndex(bufferL
ist.Count-1);
          return tmp.startPost + blocksize;
         }
          return 0;
        }
        public int getTestIgnoredSession()
        {
            return ignoredConnection;
        }
        public List<string>
getTestRequestAndHost()
        {
            return requestAndHost;
        }
        public void stopStream()
        {
            ignoredConnection = 0;
            enable = false;
            reset();
        }
        public void writeToStream(IntPtr
streamBuffer)
        {
            Marshal.Copy(bassBuffer, 0,
streamBuffer, bassBuffer.Length);
        }
    }
```

RedToHub.cs

```
using System;using System.Net;using
System.Net.Sockets;using
System.Threading;using
System.Collections.Generic;using
System.Text;using System.Linq;using
System.IO;using Db4objects.Db4o;using
Db4objects.Db4o.Linq;using
Newtonsoft.Json;using
Newtonsoft.Json.Linq;
namespace upikapik
{
    class RedToHub
    {
        IObjectContainer db;
        private string trackerIP { get;
set; }
        private int port { get; set; }
        private IPEndPoint server;
        private string comToHub;
        private string response;
        private string[] parsedCommand;
        private bool connect = true; //
potential race condition
        private int fileCnt;
        public RedToHub(string dbName,
string trackerIP, int port)
        {
            db =
Db4oEmbedded.OpenFile(dbName);
            this.trackerIP = trackerIP;
            this.port = port;
            server = new
IPEndPoint(IPAddress.Parse(trackerIP),
port);
        }
        // is connect
        public bool isConnect()
        {
            return connect;
        }
        // this is where the socket client
and thread live :-D
        public void command(string command)
        {
```

```
ThreadPool.QueueUserWorkItem(new
WaitCallback(connectionHandler));
            this.comToHub = command;
            parsedCommand =
comToHub.Split(';');
        }
        // get file list from db
        public List<file_list>
getFileList()
        {
            List<file_list> fileList = new
List<file_list>();
            dynamic files = from file_list
f in db select f;
            fileCnt = 0;
            foreach (var item in files){
                fileList.Add(item);
                fileCnt++;
            }
            return fileList;
        }
        // parse command and choose
suitable command to execute
        private void
connectionHandler(object pStateObj)
        {
            TcpClient red = new
TcpClient();
            red.SendTimeout = 3000;
            red.ReceiveTimeout = 3000;
            try
            {
                red.Connect(server);
            } catch (SocketException ex)
            {
                connect = false;
            }
            if (connect == true){
                // create buffer and set
encoding to UTF8
                byte[] buffer =
System.Text.Encoding.UTF8.GetBytes(comToHu
b);
                // prepare stream to write
or read
                NetworkStream stream =
red.GetStream();
                // send command to HUB
                if (parsedCommand[0] ==
"FL") {
                    buffer =
System.Text.Encoding.UTF8.GetBytes(parsedC
ommand[0]);
                } else if (parsedCommand[0]
== "FD") {
                    buffer =
System.Text.Encoding.UTF8.GetBytes(parsedC
ommand[0] + ";;" + parsedCommand[1]);
                } else if (parsedCommand[0]
== "UP") {
                    //
command;;id_file;;block_avail
                    buffer =
System.Text.Encoding.UTF8.GetBytes(parsedC
ommand[0] + ";;" + parsedCommand[1] + ";;"
+ parsedCommand[2]);
                } else if (parsedCommand[0]
== "NA") {
                    buffer =
System.Text.Encoding.UTF8.GetBytes(parsedC
ommand[0]);
                } else if (parsedCommand[0]
== "ADD"){
                    //
command;;file_name;;bitrate;;samplerate;;s
ize
                    buffer =
System.Text.Encoding.UTF8.GetBytes(parsedC
ommand[0] + ";;" + parsedCommand[1] + ";;"
+ parsedCommand[2] + ";;" +
parsedCommand[3] + ";;" +
parsedCommand[4]);
                } else if (parsedCommand[0]
== "UN") {
                    //
command;;file_name;;block_avail
                    buffer =
System.Text.Encoding.UTF8.GetBytes(parsedC
ommand[0] + ";;" + parsedCommand[1] + ";;"
+ parsedCommand[2]);
                }
                stream.Write(buffer, 0,
buffer.Length);
                // clear buffer
                if (parsedCommand[0] == "FL"
|| parsedCommand[0] == "FD") {
                    buffer = new Byte[1024];
                    int byteCnt =
stream.Read(buffer, 0, buffer.Length);
                    response =
System.Text.Encoding.UTF8.GetString(buffer
);
                    if (parsedCommand[0] ==
"FL")

comFileList(response);
                    if (parsedCommand[0] ==
"FD")

comFileDetail(response,
Convert.ToInt16(parsedCommand[1]));
                }
                stream.Close();
                red.Close();
            }
        }
        // send command and retrive file
list from hub
```

```
        private void comFileList(string
response)
        {
            List<file_list> list =
JsonConvert.DeserializeObject<List<file_li
st>>(response);
            dynamic all = from file_list f
in db select f;
            // delete all
            foreach (var item in all)
                db.Delete(item);
            // write all
            foreach (var item in list)
                db.Store(item);
        }
        // send command and retrieve file
detail from hub
        private void comFileDetail(string
response, int id)
        {
            string fileDetail = response;
            List<file_host_rel> update =
JsonConvert.DeserializeObject<List<file_ho
st_rel>>(response);
            dynamic all_ok = from
file_host_rel f in db select f;
            // delete all apropriate
            foreach (var item in all_ok)
                db.Delete(item);
            // insert all from response
            foreach (var item in update)
                db.Store(item);
        }
        // Add entry to file_available db
        public void addFileAvail(string
filePath, int block_avail = 0)
        {

storeFileAvailable(createInfo(filePath,
block_avail));
        }
        // block_avail = 0 is full
        public file_available
createInfo(string filePath, int
block_avail = 0)
        {
            MP3Header reader = new
MP3Header();
            file_available file = new
file_available();


reader.ReadMP3Information(filePath);
file.nama =
Path.GetFileName(Uri.UnescapeDataString(fi
lePath).Replace("/", "\\"));
            file.bitrate =
reader.intBitRate;
            file.samplerate =
reader.intFrequency;
            file.size =
Convert.ToInt32(reader.lngFileSize);
            // there are ambuguity between
block and frame, actually this is same
thing
            if (block_avail == 0){
                file.block_avail =
reader.intTotalFrame;
                file.full = true;
            }
            else {
                file.block_avail =
block_avail;
                file.full = false;
            }
             return file;
        }
        public void
storeFileAvailable(file_available file)
        {
            int fileIdInHub =
getIdFromName(file.nama);
            this.command("ADD;" + file.nama
+ ";" + file.bitrate + ";" +
file.samplerate + ";" + file.size);
            Thread.Sleep(500);
            dynamic t = from file_list sip
in db select sip;

            foreach (var item in t) {
                if (item.nama == file.nama)
                    fileIdInHub =
item.id_file;
            }
            db.Store(file);
            this.command("UN;" + file.nama
+ ";" + file.block_avail);
        }
        // we can't do this because we not
yet get the list before add it.
        public int getIdFromName(string
name)
        {
            int id_file = 0;
            dynamic f = from file_list g in
db where g.nama.Equals(name) select g;
            foreach (var item in f)
                id_file = item.id_file;
            return id_file;
        }
        public Queue<Hosts>
getAvailableHost(string nama)
        {
            Queue<Hosts> hosts = new
Queue<Hosts>();

            String strHostName =
Dns.GetHostName();
            IPHostEntry ipEntry =
```

```
Dns.GetHostEntry(strHostName);
            IPAddress[] addr =
ipEntry.AddressList;

            dynamic obj = from
file_host_rel f in db where
f.nama.Equals(nama) select f;
            foreach (var item in obj) {
                bool find = false;
                Hosts host = new Hosts();
                foreach (IPAddress address
in addr){

                    if (address.ToString()
== item.ip.ToString())
                        find = true;
                    if
(address.Equals(null))
                        break;
                }
                if(!find) {
                    host.blockAvail =
item.block_avail;
                    host.peer = new
IPEndPoint(IPAddress.Parse(item.ip),
1338);
                    hosts.Enqueue(host);
                }
            }
            return hosts;
        }
        public void updateFileInfo(int
id_file, int blockAvailable)
        {
            file_available file = new
file_available();
            file.id_file = 45;
            file.nama = "Keren";
            db.Store(file);
            dynamic f = from file_list g in
db where g.id_file.Equals(id_file) select
g;
            foreach (var item in f)  {
                file.id_file = id_file;
                file.nama = item.nama;
                file.samplerate =
item.samplerate;
                file.bitrate = item.bitrate;
                file.size = item.size;
                file.block_avail =
blockAvailable;
                if (blockAvailable ==
item.size)
                    file.full = true;
                else
                    file.full = false;
                db.Store(file);
            }
            dynamic t = from file_available
g in db select g;
        }
        public int
getBlockAvailableSize(string nama)
        {
            file_available file = new
file_available();
            int block_avail = 0;
           dynamic f = from file_available g
in db where g.nama.Equals(nama) select g;
            foreach (var item in f)
                block_avail =
item.block_avail;
            return block_avail;
        }
        public bool isFull(string nama)
        {
            bool full = false;
           dynamic f = from file_available g
in db where g.nama.Equals(nama) select g;
            foreach (var item in f)
                full = item.full;
            return full;
        }  }
}
```

BassPlayer.cs

```
using System;using
System.Collections.Generic;using
System.Timers;using System.Text;using
Un4seen.Bass;
namespace upikapik
{class BassPlayer
    {
        private int stream;
        private long streamLen; // stream
size in byte
        private long streamPos; // stream
position in byte
        private string path;
        private System.Timers.Timer
mainTime;
        private bool local = false;
        /*< initialize everything to get
bass set and ready >*/
        public BassPlayer()
        {
            // initialize bass device
            if (!(Bass.BASS_Init(-1, 44100,
BASSInit.BASS_DEVICE_DEFAULT,
IntPtr.Zero)))
                throw new
System.InvalidOperationException("Can't
initialize bass device");
            // setup timer
            mainTime = new
System.Timers.Timer(500);
            mainTime.Elapsed += new
ElapsedEventHandler(onMainTime);
```

```
            mainTime.AutoReset = true;
            mainTime.Enabled = true;
        }
        /*< set path from active stream
source>*/
        private void setPath(string path)
        {
            this.path = path;
        }
        /*< Set volume>*/
        public void setVolume(int vol)
        {
Bass.BASS_SetVolume(((float)vol) / 10);
        }
        /*< Get path from active stream
source>*/
        public string getPath()
        {
            return path;
        }
        /*< Get filename from active stream
source>*/
        public string getFileName()
        {
            return path2FileName(path);
        }
        /*< Get the length from stream
source in byte> */
        public long getLenByte()
        {
            return streamLen;
        }
        /*< Get the length from stream
source in seconds> */
        public int getLenSec()
        {
            long len =
Bass.BASS_ChannelGetLength(stream);
            return
(int)Bass.BASS_ChannelBytes2Seconds(stream
, len);
        }
        /*< Get current playback position
from stream in byte> */
        public long getPosByte()
        {
            return streamPos;
        }
        /* < Get current playback position
from stream in second>*/
        public int getPosSec()
        {
            return
(int)Bass.BASS_ChannelBytes2Seconds(stream
, streamPos);
        }
        /*< Get current volume> */
        public int getVolume()
        {
            return (int)
(Bass.BASS_GetVolume() * 10);
        }
        /*< Is the stream paused ?>*/
        public bool isPause()
        {
            if
(Bass.BASS_ChannelIsActive(stream) ==
BASSActive.BASS_ACTIVE_PAUSED)
                return true;
            else
                return false;
        }
        /*< Is the stream not active ?> */
        public bool isActive()
        {
            if
(Bass.BASS_ChannelIsActive(stream) ==
BASSActive.BASS_ACTIVE_STOPPED)
                return false;
            else
                return true;
        }
        /*< Play the stream from file> */
        public void play_local(string path)
        {
            this.setPath(path);
            local = true;
            BASSActive status;
            status =
Bass.BASS_ChannelIsActive(stream);

            if (status ==
BASSActive.BASS_ACTIVE_PLAYING)

Bass.BASS_StreamFree(stream);
            if ((stream =
Bass.BASS_StreamCreateFile(path, 0L, 0L,
BASSFlag.BASS_DEFAULT)) != 0)  {

Bass.BASS_ChannelPlay(stream, false);
                streamLen =
Bass.BASS_ChannelGetLength(stream);
            }  else
                throw new
System.InvalidOperationException("Can't
open file to play");
        }
        /* < Play the stream from buffer>*/
        public void play_buffer(IntPtr
buffer, int length)
        {
            local = false;
            if
(Bass.BASS_ChannelIsActive(stream) ==
BASSActive.BASS_ACTIVE_PLAYING)

Bass.BASS_StreamFree(stream);
            if ((stream =
```

```
Bass.BASS_StreamCreateFile(buffer, 0L,
length, BASSFlag.BASS_DEFAULT)) != 0)

Bass.BASS_ChannelPlay(stream, false);
        }
        /*< Pause or resume the stream
according to current state of stream >*/
        public void pause_resume()
        {
            if (stream != 0)  {
                // if stream playing, pause
it
                if
(Bass.BASS_ChannelIsActive(stream) ==
BASSActive.BASS_ACTIVE_PLAYING)   {

Bass.BASS_ChannelPause(stream);
                    mainTime.Stop();
                }
                // if stream paused, play it
                else if
(Bass.BASS_ChannelIsActive(stream) ==
BASSActive.BASS_ACTIVE_PAUSED){
                    mainTime.Start();

Bass.BASS_ChannelPlay(stream, false);
                } }
        }
        /*< Stop the stream >*/
        public void stop()
        {
            Bass.BASS_ChannelStop(stream);
        }
        /*< Seek the stream to certain
position > */
        public void seek(int pos)
        {
            if (local)

Bass.BASS_ChannelSetPosition(stream,
(double)pos);
            else

Bass.BASS_ChannelSetPosition(stream,
(double)pos);
        }
        /*< Free resource used by bass > */
        public void free()
        {
            Bass.BASS_Free();
        }
        /*< Set the maximum limit to play,
the limit is from available block >*/
        public void setLimit(int limit)
        {

        }
        /*< Take filename from complete
path > */
        private string path2FileName(string
path)
        {
            return
System.IO.Path.GetFileName(path);
        }
        /*< timer to maintain dirty job :p
> */
        private void onMainTime(object
source, ElapsedEventArgs e)
        {
            streamPos =
Bass.BASS_ChannelGetPosition(stream);
        }
    }
}
```

DB.cs

```
using System;using
System.Collections.Generic;using
System.Net.Sockets;using System.Net;using
System.Threading;using System.Text;using
System.IO;
namespace upikapik
{
    class file_host_rel
    {
        public string nama { get; set; }
        public string ip { get; set; }
        public int block_avail { get; set;
}
    }
    class file_list
    {
        public int id_file { get; set; }
        public string nama { get; set; }
        public int bitrate { get; set; }
        public int samplerate { get; set; }
        public int size { get; set; }
    }
    class file_available
    {
        public int id_file; // different
with hub numbering
        public string nama { get; set; }
        public int bitrate { get; set; }
        public int samplerate { get; set; }
        public int size { get; set; }
        public int block_avail { get; set;
}
        public bool full { get; set; }
        public bool isFull()
        {
            // if framelength*block
available + header == size this is true
            return true;
        }
    }
    class config
```

```
    {
        string trackerIp { get; set; }
    }
    class block_req
    {
        public bool is_finnish;
        public byte[] buffer;
        public int start_post;
        public block_req(bool is_finnish,
int buffer_size, int start_post)
        {
            this.is_finnish = is_finnish;
            buffer = new byte[buffer_size];
            this.start_post = start_post;
        }
    }
    class RequestProp
    {
        public IPEndPoint peer;
        public TcpClient client;
        public NetworkStream stream;
        public string filename;
        // it must be adjusted, because
file seeking add 1 automatically but
socket reading nope
        // Adjust by subtract it with 1
when write to file
        public int startPost;
        public int blockSize;
        public byte[] receiveBuffer;
    }
    class Hosts
    {
        public IPEndPoint peer;
        public int blockAvail;
    }
}
```

MainForm.cs

```
using System;using System.Text;using
System.Windows.Forms;using
System.Net.Sockets;
using System.IO;using
System.Collections.Generic;using
System.Runtime.InteropServices;using
System.Configuration;using
System.Collections;using System.Linq;
namespace upikapik
{    public partial class mainForm : Form
    {
        private const int MAX_FILE_AVAIL =
10;

        BassPlayer _player;
        private OpenFileDialog _opnFile;
        string _appDir;
        bool local = false;

        // streaming things
        byte[] buffer;
        GCHandle BufferHandler;

        // bass player things
        file_list _current_file;
        int _timeTotal;
        int _timeCurrent;
        int _indexOfPlayedFile;
        bool _shuffle = false;
        List<file_list> playList = new
List<file_list>();
        Random _rand = new Random();

        int frame_size = 0;
        int lastStartpost = 0;
        int available_sec = 0;
        int block_size = 0;

        // timer
        Timer _timerPlayer;
        Timer _timerRed; // May can be if
implemented in redToHub
        int intervalOne;
        int intervalTwo;

        // another HMR component
        RedToHub _toHub;
        AsynchRedServ _server;
        System.Threading.Thread tRedServ;
        AsynchRedStream _redStream = new
AsynchRedStream();

        public mainForm()
        {
            string myIP =
ConfigurationSettings.AppSettings["IP_HOST
"].ToString();
            string hubIP =
ConfigurationSettings.AppSettings["IP_HUB"
].ToString();
            _toHub = new
RedToHub("RedDb.db4o", hubIP, 1337);
            _server = new
AsynchRedServ(myIP, 1338);

            InitializeComponent();
            _opnFile = new
OpenFileDialog();
            _player = new BassPlayer();
            _timerPlayer = new Timer();
            _timerRed = new Timer();

            _opnFile.Filter = "mp3 (*.mp3)|
*.mp3";
            _opnFile.Multiselect = false;

            _timerPlayer.Interval = 100;
            _timerPlayer.Tick += new
EventHandler(onTimerPlayer);
            _timerRed.Interval = 540000; //
```

```
9 minutes
            _timerRed.Tick += new
EventHandler(onTimerRed);
            _timerRed.Start();
            _toHub.command("NA");
            tRedServ = new
System.Threading.Thread(() => {
_server.startListening(); });
            tRedServ.Start();

            _appDir =
Directory.GetCurrentDirectory();
        }
        private void btnPlay_Click(object
sender, EventArgs e)
        {
            _player.pause_resume();
        }
        private void btnOpen_Click(object
sender, EventArgs e)
        {
            if (_toHub.howMuch() <
MAX_FILE_AVAIL)  {
                if (_opnFile.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)  {
                    bool fail = false;
                    string path =
_opnFile.FileName;
                    string filename =
_opnFile.SafeFileName;
                    try  {

System.IO.File.Copy(path,
Path.Combine(_appDir, "music", filename));
                    }  catch (Exception ex)
{

MessageBox.Show(ex.Message, "Failed to
copy file");
                        fail = true;
                    }
                    if (!fail)

_toHub.addFileAvail(path, 0);
                }
            }  else
                MessageBox.Show("File
available in network is reached maximum
limit");
        }
        private void barSeek_Scroll(object
sender, EventArgs e)
        {
            _player.seek(barSeek.Value);
        }
        private void
listPlay_DoubleClick(object sender,
EventArgs e)
        {
            if (listPlay.Items.Count != 0)
{
                _indexOfPlayedFile =
listPlay.SelectedIndex;
                // set current file
                _current_file =
playList.Find(p =>
p.nama.Equals(listPlay.SelectedItem));
                play();
                _timerPlayer.Start();
            }
        }
        private void onTimerPlayer(object
source, EventArgs e)
        {
            intervalOne++;
            intervalTwo++;

            _timeCurrent =
_player.getPosSec();
            lblStatus.Text = "Time : " +
s2t(_timeTotal) + " / " +
s2t(_player.getPosSec());

            if (!
(_redStream.getLastValueOfBuffer() == 0))
{
                lastStartpost =
_redStream.getLastValueOfBuffer();
                barProgress.Value = (int)
(((float)lastStartpost + block_size) /
(float)_current_file.size * 100);
            }
            if (_timeCurrent != -1)
                barSeek.Value =
_timeCurrent;
            if (intervalOne == 10)  {
                if (!local)  {

_player.play_buffer(BufferHandler.AddrOfPi
nnedObject(), _current_file.size);
                    _timeTotal =
_player.getLenSec();
                    barSeek.SetRange(0,
_timeTotal);
                }
            }
            if (intervalTwo == 40)   {
                // update to HUB and file
avail db
                int available =
lastStartpost/frame_size;

_toHub.updateFileInfo(_current_file.id_fil
e, available);

_toHub.command("UN;"+_current_file.nama+";
"+available);
                intervalTwo = 0;
            }
            //write to buffer
```

```
        if(!local)
_redStream.writeToStream(BufferHandler.AddrOfPinnedObject());
        }
        private void onTimerRed(object source, EventArgs e)
        {
            _toHub.command("FL");
            _toHub.command("NA");
            refreshList();
        }
        private TimeSpan s2t(int seconds)
        {
            TimeSpan time = new TimeSpan(0, 0, seconds);
            return time;
        }
        private void barVol_Scroll(object sender, EventArgs e)
        {
_player.setVolume(barVol.Value);
        }
        private void play()
        {
            _player.stop();
            if (_toHub.isFull(_current_file.nama)) {
                local = true;
                _player.play_local("music\\" + _current_file.nama);
                _timeTotal = _player.getLenSec();
                barSeek.SetRange(0, _timeTotal);
            }   else  {
                local = false;
                intervalOne = 0;
                //clean the stream properties
                _redStream.stopStream();
                _redStream.closeFile();

                //clean buffer
                try
                    BufferHandler.Free();
                catch (Exception ex)
                {

                }
                //get host info before playing
                _toHub.command("FD;" + _current_file.id_file);
System.Threading.Thread.Sleep(200);
_redStream.getHostsAvail(_toHub.getAvailableHost(_current_file.nama));
                //start stream
_redStream.startStream(_current_file.id_file, _current_file);

                //get streaming properties
                block_size = _redStream.getBlockSize();
                frame_size = _redStream.getFrameSize();
                //_timeTotal = (int) ((_current_file.size / frame_size) * 0.026);

                //set buffer
                buffer = new byte[_current_file.size];
                BufferHandler = GCHandle.Alloc(buffer, GCHandleType.Pinned);

            }
            //barProgress.Value = 0;
            this.Text = "UpikApik : " + _current_file.nama;
            barVol.Value = _player.getVolume();
        }
        private void checkBox1_CheckedChanged(object sender, EventArgs e)
        {
            if (_shuffle == false)
                _shuffle = true;
            else
                _shuffle = false;
        }
        private void btnRefresh_Click(object sender, EventArgs e)
        {
            _toHub.command("FL");
System.Threading.Thread.Sleep(100);
            refreshList();
        }
        private void refreshList()
        {
            playList = _toHub.getFileList();
            listPlay.Items.Clear();
            foreach (var item in playList)
listPlay.Items.Add(item.nama);
        }
        private void mainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
```

```
            _server.enable = false;
((IDisposable)_server).Dispose();
        }
        private void btnStop_Click(object
sender, EventArgs e)
        {
            _player.stop();
            barSeek.Value = 0;
            _timerPlayer.Stop();
            lblStatus.Text = "Time :
00:00 / 00:00";
            if (!local){
                _redStream.stopStream();
                _redStream.closeFile();
            }
        }
        private void btnTest_Click(object
sender, EventArgs e)
        {
            int totalDelay = 0; // belum
implement
            int totalIgnored =
_redStream.getTestIgnoredSession();
            List<string> reqAndHost =
_redStream.getTestRequestAndHost();
            var q = from x in reqAndHost
                    group x by x into g
                    let count = g.Count()
                    orderby count descending
                    select new { Value =
g.Key, Count = count };
            string hostReport = "\n";
            foreach (var item in q)
                hostReport = hostReport +
"\n" + item.Value + " : " + item.Count;
            string message =
                "File name  = " +
_current_file.nama + "\n" +
                "File size  = " +
_current_file.size + " byte " + "\n" +
                "Bitrate    = " +
_current_file.bitrate + " Kbps" + "\n" +
                "Samplerate = " +
_current_file.samplerate + " KHz" + "\n" +
                "============= " + "\n" +
                "Total delay     = " +
totalDelay + "\n" +
                "Ignored Attempt = " +
totalIgnored + "\n" +
                "Host requested" +
hostReport;
            MessageBox.Show(message);
        }
    }
}
```