private class AcceptThread extends Thread {

private final BluetoothServerSocket

mmServerSocket;

public AcceptThread() {

// Use a temporary object that is later assigned to mmServerSocket,

// because mmServerSocket is final

BluetoothServerSocket tmp = null;

try {

// MY\_UUID is the app's UUID string, also used by the client code

tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, MY\_UUID);

} catch (IOException e) { }

mmServerSocket = tmp;

}

public void run() {

BluetoothSocket socket = null;

// Keep listening until exception occurs or a socket is returned

while (true) {

try {

socket = mmServerSocket.accept();

} catch (IOException e) {

break;

}

// If a connection was accepted

if (socket != null) {

// Do work to manage the connection (in a separate thread)

manageConnectedSocket(socket);

mmServerSocket.close();

break;

}

}

}

/\*\* Will cancel the listening socket, and cause the thread to finish \*/

public void cancel() {

try {

mmServerSocket.close();

} catch (IOException e) { }

}

}

private class ConnectThread extends Thread {

private final BluetoothSocket mmSocket;

private final BluetoothDevice mmDevice;

public ConnectThread(BluetoothDevice device) {

// Use a temporary object that is later assigned to mmSocket,

// because mmSocket is final

BluetoothSocket tmp = null;

mmDevice = device;

// Get a BluetoothSocket to connect with the given BluetoothDevice

try {

// MY\_UUID is the app's UUID string, also used by the server code

tmp = device.createRfcommSocketToServiceRecord(MY\_UUID);

} catch (IOException e) { }

mmSocket = tmp;

}

public void run() {

// Cancel discovery because it will slow down the connection

mBluetoothAdapter.cancelDiscovery();

try {

// Connect the device through the socket. This will block

// until it succeeds or throws an exception

mmSocket.connect();

} catch (IOException connectException) {

// Unable to connect; close the socket and get out

try {

mmSocket.close();

} catch (IOException closeException) { }

return;

}

// Do work to manage the connection (in a separate thread)

manageConnectedSocket(mmSocket);

}

/\*\* Will cancel an in-progress connection, and close the socket \*/

public void cancel() {

try {

mmSocket.close();

} catch (IOException e) { }

}

}

private class ConnectedThread extends Thread {

private final BluetoothSocket mmSocket;

private final InputStream mmInStream;

private final OutputStream mmOutStream;

public ConnectedThread(BluetoothSocket socket) {

mmSocket = socket;

InputStream tmpIn = null;

OutputStream tmpOut = null;

// Get the input and output streams, using temp objects because

// member streams are final

try {

tmpIn = socket.getInputStream();

tmpOut = socket.getOutputStream();

} catch (IOException e) { }

mmInStream = tmpIn;

mmOutStream = tmpOut;

}

public void run() {

byte[] buffer = new byte[1024]; // buffer store for the stream

int bytes; // bytes returned from read()

// Keep listening to the InputStream until an exception occurs

while (true) {

try {

// Read from the InputStream

bytes = mmInStream.read(buffer);

// Send the obtained bytes to the UI activity

mHandler.obtainMessage(MESSAGE\_READ, bytes, -1, buffer)

.sendToTarget();

} catch (IOException e) {

break;

}

}

}

/\* Call this from the main activity to send data to the remote device \*/

public void write(byte[] bytes) {

try {

mmOutStream.write(bytes);

} catch (IOException e) { }

}

/\* Call this from the main activity to shutdown the connection \*/

public void cancel() {

try {

mmSocket.close();

} catch (IOException e) { }

}

}

public class BTComm extends Thread {

BluetoothSocket socket;

MessageReceiver onMessageReceiverListener;

boolean run = false;

protected void sendMessage(String message){ //küldés

if(socket != null){

try {

OutputStream os = socket.getOutputStream();

os.write(message.getBytes());

os.flush();

} catch (IOException e) {

e.printStackTrace();

}

}

}

protected void receiveMessage(){ //fogadás

byte[] buffer = new byte[1024];

int len;

while(run){

if(socket != null){

try {

// Kiolvas (feltételezzük, hogy nem hosszabb 1024)

len = socket.getInputStream().read(buffer);

String message = new String(buffer, 0, len);

if(onMessageReceiverListener != null) // értesítés

onMessageReceiverListener.onMessageReceived(message);

} catch (IOException e) {

e.printStackTrace();

closeSocket();

}

}

}

}

public void closeSocket(){

this.run = false;

if(socket != null){

try {

socket.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

public void setOnMessageReceiverListener(MessageReceiver onMessageReceiverListener) {

this.onMessageReceiverListener = onMessageReceiverListener;

} // Setter & Getters

interface MessageReceiver{ // Interfész a beérkező üzenetértesítéshez

void onMessageReceived(String message);

}

}

public class BTServer extends BTComm {

BluetoothServerSocket bss; // Ez még kimondottan server socket

public BTServer(BluetoothAdapter adapter, String name, UUID uuid) {

try {

this.bss = adapter.listenUsingRfcommWithServiceRecord(name, uuid);

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

this.run = true;

while(run){

try {

this.socket = bss.accept(); // itt lehetne timeout-ot adni is

Log.d("bt", "Kliens megerkezett.");

receiveMessage(); // megvan az aktív socket, kezdünk vele valamit (szerver oldal)

} catch (IOException e) {

e.printStackTrace();

}

}

super.run();

}

// Míg kliens odalról elég a socket-et zárni, addig itt a server socket-et is ajánlott

@Override

public void closeSocket() {

super.closeSocket(); // ős meghívása, sima socket zárása

try {

bss.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

public class BTClient extends BTComm {

public BTClient(BluetoothDevice device, UUID uuid) {

try {

this.socket = device.createRfcommSocketToServiceRecord(uuid);

} catch (IOException e) {

e.printStackTrace();

}

}

@Override

public void run() {

try {

socket.connect();

Log.d("bt", "Kliens kapcsolódott a távoli szerverhez.");

run = true;

super.receiveMessage(); // megvan az aktív socket, kezdünk vele

// valamit (kliens oldal)

} catch (IOException e) {

e.printStackTrace();

closeSocket();

}

super.run();

}

}