Приложение 1

ПРОГРАММНЫЙ МОДУЛЬ ВЗАИМОДЕЙСТВИЯ КАССОВОГО И БАНКОВСКОГО ПРИЛОЖЕНИЙ НА POS-ТЕРМИНАЛЕ

ПМ ВКБП

Оглавление

[1. Droidbase 3](#_Toc515453765)

[1.1. Файл application/service/TtkBackgroundService.java 3](#_Toc515453766)

[1.2. Файл application/service/TTKResponseHelper.java 9](#_Toc515453767)

[1.3. Файл helpers/ttk/TtkJsonProtocol.java 12](#_Toc515453768)

# Droidbase

## 1.1. Файл application/service/TtkBackgroundService.java

public class TtkBackgroundService extends Service

{

private final LogChannel log\_;

private NotificationManager notification\_manager\_;

private TTKThread ttk\_thread\_;

private final int NOTIFICATION\_ID = 1;

private final int PORT = 43888;

private final static int timeout = 6000;

//========================================================================

public TtkBackgroundService()

{

super();

log\_ = new LogChannel("TTKServ");

}

//========================================================================

public static class ServiceLoader extends BroadcastReceiver

{

public void onReceive(Context context, Intent intent)

{

context.startService(new Intent(context,TtkBackgroundService.class));

}

}

//========================================================================

@Override public void onCreate()

{

log\_.logInfo("TTK Service onCreate start");

super.onCreate();

notification\_manager\_ =

(NotificationManager)getSystemService(NOTIFICATION\_SERVICE);

log\_.logInfo("TTK Service onCreate end");

}

//========================================================================

@Override public int onStartCommand(Intent intent, int flags, int startId)

{

log\_.logInfo("TTK Service onStartCommand start");

if (ttk\_thread\_ == null)

{

createNotification();

runTTKThread(PORT);

}

log\_.logInfo("TTK Service onStartCommand end");

return START\_STICKY;

}

//========================================================================

private void createNotification()

{

NotificationCompat.Builder builder =

new NotificationCompat.Builder(this, "");

builder.setOngoing(true)

.setContentTitle(getResources().getString(R.string.app\_name))

.setSmallIcon(R.mipmap.posterminal)

.setContentText(getResources().getString(

R.string.ttk\_notification\_text));

startForeground(NOTIFICATION\_ID, builder.build());

}

//========================================================================

private void runTTKThread(int port)

{

log\_.logInfo("creating and starting ttk thread...");

ttk\_thread\_ = new TTKThread(port);

ttk\_thread\_.start();

log\_.logInfo("ttk thread started");

}

//========================================================================

@Override public void onDestroy()

{

log\_.logInfo("ttk service onDestroy start");

if (ttk\_thread\_ != null)

{

ttk\_thread\_.cancel();

ttk\_thread\_.interrupt();

log\_.logInfo("waiting fot ttk thread to stop");

while (!ThreadUtils.joinThread(ttk\_thread\_))

{

log\_.logInfo("interrupt try on ttk thread finish waiting");

ThreadUtils.sleep(100);

}

log\_.logInfo("ttk thread joining finished");

}

log\_.logInfo("canceling notification");

notification\_manager\_.cancel(NOTIFICATION\_ID);

stopSelf();

super.onDestroy();

log\_.logInfo("ttk service onDestroy end");

}

//========================================================================

@Override public IBinder onBind(Intent intent)

{

return null;

}

//========================================================================

private class TTKThread extends Thread

{

private final int port\_;

private final AtomicBoolean cancel\_requested\_ = new

AtomicBoolean(false);

ServerSocket server\_socket\_;

TTKServerChannel curr\_channel\_;

//======================================================================

public TTKThread(int port)

{

log\_.logInfo("ttk thread created on port " + port);

port\_ = port;

cancel\_requested\_.set(false);

}

public void cancel()

{

cancel\_requested\_.set(true);

}

public boolean isCancelled()

{

return cancel\_requested\_.get();

}

//======================================================================

@Override public void run()

{

try

{

doRun();

}

finally

{

log\_.logInfo("ttk thread end point");

}

}

//======================================================================

private void doRun()

{

log\_.logInfo("ttk thread doRun start");

if (!createServerSocket())

{

log\_.logError("failed to create ttk server socket");

return;

}

try

{

while (true)

{

if (isCancelled())

{

log\_.logInfo("ttk thread canceled before accept");

break;

}

Socket client\_socket = null;

try

{

client\_socket = server\_socket\_.accept();

}

catch (SocketTimeoutException e)

{

continue;

}

catch (Exception e)

{

log\_.logInfo("exception during accept");

e.printStackTrace();

continue;

}

if (client\_socket != null)

{

log\_.logInfo("ttk client connection accepted");

try

{

processClientConnection(client\_socket);

}

catch (IOException e)

{

log\_.logInfo("io exception during processing client

connection");

e.printStackTrace();

}

finally

{

closeClientSocket(client\_socket);

}

}

}

}

finally

{

log\_.logInfo("ttk thread main cycle finished");

closeAllSockets();

}

}

//======================================================================

private boolean checkRequest(TTKMessage request)

{

if (request == null)

{

return false;

}

if (request.getMessageID() == null)

{

log\_.logError("process: MSGID is missing");

return false;

}

if (request.getECRNumber() == null)

{

log\_.logError("process: ECRNumber is missing");

return false;

}

if (request.getERN() == null)

{

log\_.logError("process: ERN is missing");

return false;

}

return true;

}

//======================================================================

private boolean sendWrongFormatResponse(TTKTransport ttk\_transp,

TTKMessage request)

{

try

{

log\_.logInfo("sending wrong format response");

ttk\_transp.sendMessage(TTKResponseHelper.

makeMessageWrongFormatResponse(request));

log\_.logInfo("wrong format response sent");

return true;

}

catch (IOException e)

{

log\_.logError("send failed: connection lost?");

return false;

}

catch (TTKFormatException fe)

{

log\_.logError("send failed: error in TTK message packet");

return false;

}

}

//======================================================================

private boolean sendUnknownCommandResponse(TTKTransport ttk\_transp,

TTKMessage request)

{

try

{

log\_.logInfo("sending unknown command response");

ttk\_transp.sendMessage(TTKResponseHelper.

makeMessageUnknownCommandResponse(request));

log\_.logInfo("unknown command response sent");

return true;

}

catch (IOException e)

{

log\_.logError("send failed: connection lost?");

return false;

}

catch (TTKFormatException fe)

{

log\_.logError("send failed: error in TTK message packet");

return false;

}

}

//======================================================================

private boolean createServerSocket()

{

if (server\_socket\_ != null)

{

log\_.logWarn("server socket already created");

return false;

}

try

{

final ServerSocket new\_socket = new ServerSocket(port\_);

new\_socket.setSoTimeout(1000);

server\_socket\_ = new\_socket;

return true;

}

catch (IOException e)

{

e.printStackTrace();

return false;

}

}

//======================================================================

private void closeAllSockets()

{

log\_.logInfo("closing ttk sockets");

closeServerSocket();

}

//======================================================================

private void closeClientSocket(Socket socket)

{

if (socket == null)

return;

try

{

log\_.logInfo("closing client socket");

socket.close();

log\_.logInfo("client socket closed");

}

catch (IOException e)

{

log\_.logInfo("client socket closing exception");

e.printStackTrace();

}

}

//======================================================================

private void closeServerSocket()

{

if (server\_socket\_ == null)

{

return;

}

try

{

log\_.logInfo("closing server socket");

server\_socket\_.close();

log\_.logInfo("server socket closed");

}

catch (IOException e)

{

log\_.logInfo("server socket closing exception");

e.printStackTrace();

}

}

}

}

## 1.2. Файл application/service/TTKResponseHelper.java

public class TTKResponseHelper

{

static public final String RESPONSE\_CODE\_WRONG\_FORMAT = "FE";

static public final String RESPONSE\_CODE\_UNKNOWN\_COMMAND = "UN";

static public final String RESPONSE\_CODE\_NOT\_FOUND = "NF";

static public final String RESPONSE\_CODE\_UNKNOWN\_RESPONSE = "UR";

static public final String RESPONSE\_CODE\_NOT\_INITIALIZED = "NI";

static public final String RESPONSE\_CODE\_APP\_ERROR = "ER";

static public final String RESPONSE\_CODE\_BAD\_ERN = "B4";

static public final String RESPONSE\_CANCELED = "";

static public final String RESPONSE\_CODE\_GENERIC\_SUCCESS = "00";

//========================================================================

public static TTKTransactionMessage makeMessageApprovedResponse(TTKMessage

request)

{

return makeMessageResponse(true, RESPONSE\_CODE\_GENERIC\_SUCCESS,

request);

}

//========================================================================public static TTKTransactionMessage makeMessageWrongFormatResponse

(TTKMessage request)

{

return makeMessageResponse(false, RESPONSE\_CODE\_WRONG\_FORMAT, request);

}

//========================================================================public static TTKTransactionMessage makeMessageAppErrorResponse(TTKMessage

request)

{

return makeMessageResponse(false, RESPONSE\_CODE\_APP\_ERROR, request);

}

//========================================================================

public static TTKTransactionMessage makeMessageUnknownCommandResponse

(TTKMessage request)

{

return makeMessageResponse(false, RESPONSE\_CODE\_UNKNOWN\_COMMAND,

request);

}

//========================================================================

public static TTKTransactionMessage makeMessageNotFoundResponse(TTKMessage

request)

{

return makeMessageResponse(false, RESPONSE\_CODE\_NOT\_FOUND, request);

}

//========================================================================

public static TTKTransactionMessage makeMessageBadERNResponse(TTKMessage

request)

{

return makeMessageResponse(false, RESPONSE\_CODE\_BAD\_ERN, request);

}

//========================================================================

public static TTKTransactionMessage makeMessageResponse(boolean success,

String response\_code, TTKMessage request)

{

if (request == null)

{

return null;

}

final TTKTransactionMessage response = new

TTKTransactionMessage(request);

response.setApprove(makeApproveCode(success));

response.setResponseCode(response\_code);

return response;

}

//========================================================================

private static char makeApproveCode(boolean success)

{

return success ? 'Y' : 'N';

}

//========================================================================

private static void fillResponseWithContext(TTKTransactionMessage msg,

OperationContext ctx) throws ConvertorException

{

msg.setCurrencyAlpha(ctx.getCurrencyName());

msg.setTransactionAmount(ctx.getTransactionAmount());

msg.setCurrencyDecimalPoint(2);

final String masked\_PAN = ctx.getMaskedPAN();

if (masked\_PAN != null)

{

msg.setPAN(masked\_PAN);

}

msg.setInvoiceNumber(ctx.getInvoiceNumber());

msg.setAuthorizationID(ctx.getAuthCode());

msg.setDatetime(ctx.getTrDateTime());

msg.setTerminalId(Terminal.getTerminalInfo().terminal\_id);

msg.setMerchantNo(Terminal.getTerminalInfo().merchant\_id);

msg.setPOSEntryMode(buildTTKPOSEntryMode(ctx.getCardInputInfo(),

ctx.getOnline()));

msg.setPOSConditionCode(ctx.getPosConditionCode());

msg.setCardholderVerificationCharacter(getCVCType(ctx));

msg.setIssuerName(ctx.getCardIssuerName());

if (ctx.getRRN() != null)

{

msg.setRRN(ctx.getRRN());

}

if (ctx.getAIDHEX() != null)

{

msg.setApplicationID(Convertors.hexStr2bytes(ctx.getAIDHEX()));

}

{

final String app\_label = ctx.getAIDLable();

if (app\_label != null && !app\_label.isEmpty())

msg.setApplicationLabel(app\_label);

}

msg.setTVR(ctx.getTVR());

}

//========================================================================

private static String buildCardInputInfoCode(CardInputInfo input\_info)

{

if (input\_info == null)

return "XX";

if (input\_info.is\_cntless\_card\_enter && input\_info.is\_scr\_card\_enter)

return "07";

if (input\_info.is\_cntless\_card\_enter && input\_info.is\_msr\_card\_enter)

return "91";

if (input\_info.is\_scr\_card\_enter && input\_info.is\_fall\_back)

return "80";

if (input\_info.is\_scr\_card\_enter)

return "05";

if (input\_info.is\_msr\_card\_enter)

return "90";

if (input\_info.is\_manual\_enter)

return "01";

return "XX";

}

//========================================================================

public static boolean isAcceptedCase(OperationResult.TransactionDecision

desicion)

{

switch (desicion)

{

case Accepted:

return true;

default:

return false;

}

}

//========================================================================

private static String buildTTKPOSEntryMode(CardInputInfo input\_info,

boolean is\_online)

{

return buildCardInputInfoCode(input\_info) +

buildIsOnlineCode(is\_online);

}

}

## 1.3. Файл helpers/ttk/TtkJsonProtocol.java

public class TTKJSONProtocol extends TTKProtocol

{

public final String kTTK\_VERSION\_FIELD = "ttk\_version";

public static HashMap<TTKProtocol.RequestFields, String> request\_tags =

new HashMap<>();

public static HashMap<TTKProtocol.ResponseFields, String> response\_tags =

new HashMap<>();

private static LogChannel log\_ = new LogChannel("TtkJsonProtocol");

static

{

request\_tags.put(RequestFields.MessageID, "message\_id");

request\_tags.put(RequestFields.ECRNumber, "ecr");

request\_tags.put(RequestFields.ERN, "ern"); request\_tags.put(RequestFields.TransactionAmount, "transaction\_amount"); request\_tags.put(RequestFields.TransactionMode, "transaction\_mode"); request\_tags.put(RequestFields.PAN, "pan"); request\_tags.put(RequestFields.InvoiceNumber, "invoice\_number"); request\_tags.put(RequestFields.AuthorizationID, "authorization\_id"); request\_tags.put(RequestFields.OrderDetails, "order\_details"); request\_tags.put(RequestFields.CommodityCode, "commodity\_code"); request\_tags.put(RequestFields.RRN, "rrn"); request\_tags.put(RequestFields.SRVSubFunction, "srv\_subfunction"); request\_tags.put(RequestFields.Currency, "currency"); request\_tags.put(RequestFields.InterfaceLanguage, "interface\_language"); request\_tags.put(RequestFields.ReaderCommand, "reader\_command"); request\_tags.put(RequestFields.LocalTime, "local\_time"); request\_tags.put(RequestFields.InputCode, "input\_code"); request\_tags.put(RequestFields.InputData, "input\_data"); request\_tags.put(RequestFields.ExternalTrack2, "track2");

request\_tags.put(RequestFields.ExternalPIN, "pin");

request\_tags.put(RequestFields.HostConfig, "host\_config");

response\_tags.put(ResponseFields.MessageID, "message\_id");

response\_tags.put(ResponseFields.ECRNumber, "ecr");

response\_tags.put(ResponseFields.ERN, "ern");

response\_tags.put(ResponseFields.ResponseCode, "response\_code");

response\_tags.put(ResponseFields.TransactionAmount,

"transaction\_amount");

response\_tags.put(ResponseFields.PAN, "pan");

response\_tags.put(ResponseFields.InvoiceNumber, "invoice\_number");

response\_tags.put(ResponseFields.AuthorizationID, "authorization\_id");

response\_tags.put(ResponseFields.Date, "date");

response\_tags.put(ResponseFields.Time, "time");

response\_tags.put(ResponseFields.IssuerName, "issuer\_name");

response\_tags.put(ResponseFields.MerchantNo, "merchant\_no");

response\_tags.put(ResponseFields.ProcessingCode, "processing\_code");

response\_tags.put(ResponseFields.POSEntryMode, "pos\_entry\_mode" );

response\_tags.put(ResponseFields.POSConditionCode,

"pos\_condition\_code");

response\_tags.put(ResponseFields.CardholderVerificationCharacter,"cvc");

response\_tags.put(ResponseFields.TVR, "tvr");

response\_tags.put(ResponseFields.RRN, "rrn");

response\_tags.put(ResponseFields.BatchNo, "batch\_no");

response\_tags.put(ResponseFields.Receipt, "receipt");

response\_tags.put(ResponseFields.TerminalID, "terminal\_id");

response\_tags.put(ResponseFields.ReceiptPDS, "receipt\_pds");

response\_tags.put(ResponseFields.ReceiptSecondPDS, "receipt\_pds2");

response\_tags.put(ResponseFields.ApplicationID, "application\_id");

response\_tags.put(ResponseFields.TC, "tc");

response\_tags.put(ResponseFields.VisualHostResponse,

"visual\_host\_response");

response\_tags.put(ResponseFields.Approve, "approve");

response\_tags.put(ResponseFields.TransactionAmount2,

"transaction\_amount2");

response\_tags.put(ResponseFields.ApplicationLabel, "application\_label");

response\_tags.put(ResponseFields.CurrencyAlpha, "currency\_alpha");

response\_tags.put(ResponseFields.CurrencyDecimalPoint,

"currency\_decimal\_point");

response\_tags.put(ResponseFields.ReaderCommandResult,

"reader\_cmd\_result");

response\_tags.put(ResponseFields.LocalTime, "local\_time");

response\_tags.put(ResponseFields.StatusID, "status\_id");

response\_tags.put(ResponseFields.StatusText, "status\_text");

response\_tags.put(ResponseFields.InputTemplateID, "input\_template\_id");

response\_tags.put(ResponseFields.InputMinLength, "input\_min\_length");

response\_tags.put(ResponseFields.InputMaxLength, "input\_max\_length");

response\_tags.put(ResponseFields.InputFormat, "input\_format");

response\_tags.put(ResponseFields.InputCaption, "input\_caption");

response\_tags.put(ResponseFields.InputText, "input\_text");

response\_tags.put(ResponseFields.InputDefaultValue,

"input\_default\_value");

response\_tags.put(ResponseFields.InputCurrency, "input\_currency");

response\_tags.put(ResponseFields.InputDecimalPoint,

"input\_decimal\_point");

response\_tags.put(ResponseFields.InputItems, "input\_items");

response\_tags.put(ResponseFields.Timeout, "timeout");

}

static String getResponseTag(ResponseFields f)

{

return response\_tags.get(f);

}

static String getRequestTag(RequestFields f)

{

return request\_tags.get(f);

}

@Override

public byte[] buildHeader()

{

return "JSON".getBytes();

}

interface TTKSimpleEncoder

{

Object encode(TTKMessage msg) throws JSONException, TTKFormatException;

}

interface TTKJsonObjectEncoder

{

JSONObject encode(TTKMessage msg);

}

interface TTKDecoder

{

void decode(TTKMessage msg, Object data) throws TTKFormatException;

}

private static HashMap<TTKProtocol.ResponseFields, TTKSimpleEncoder>

tag\_simple\_encoders = new HashMap<>();

private static HashMap<TTKProtocol.ResponseFields, TTKSimpleEncoder>

tag\_object\_encoders = new HashMap<>();

private static HashMap<TTKProtocol.RequestFields, TTKDecoder>

tag\_decoders = new HashMap<>();

@Override

public byte[] encode(TTKMessage msg) throws TTKFormatException

{

JSONObject json = new JSONObject();

try

{

for (ResponseFields f : msg.getResponseFields())

{

TTKSimpleEncoder encoder = tag\_simple\_encoders.get(f);

String key = response\_tags.get(f);

if (encoder == null)

{

Log.e("TTKEncode", "no encoder for requested field: " +

f.toString());

}

Object val = encoder.encode(msg);

if (val == null)

{

continue;

}

Type type = val.getClass();

if (type.equals(Integer.class))

json.put(key, (int) val);

else if (type.equals(Long.class))

json.put(key, (long) val);

else if (type.equals(String.class))

json.put(key, val);

else if (type.equals(JSONObject.class))

json.put(key, (JSONObject) val);

else if (type.equals(byte[].class))

json.put(key, Convertors.bytes2HexStr((byte[]) val));

else if (type.equals(Character.class))

{

json.put(key, val);

}

else

{

Log.e("TTKJsonEncode", "No conversion for type [" +

type.toString() + "] for field " + key);

return null;

}

Log.d("TTKJsonEncode", "ret class is " + type.toString());

Log.i("TTKMessage", "Encoded field: " + f.toString() + "[" + key +

"]" + " Value: " + String.valueOf(val));

}

json.put(kTTK\_VERSION\_FIELD, msg.getTTKVersion());

}

catch (JSONException | TTKFormatException e)

{

throw new TTKFormatException(e);

}

return json.toString().getBytes();

}

@Override

public TTKMessage decode(byte[] msg) throws TTKFormatException

{

JSONObject json;

json = parseJson(new String(msg));

if (json == null)

{

Log.e("TTKJsoneDecode", "Bad JSON data to parse!");

return null;

}

int ttk\_ver = getIntOrDefault(json, kTTK\_VERSION\_FIELD, 1);

String msgidString = getStringOrDefault(json, request\_tags.

get(RequestFields.MessageID), null);

if (msgidString == null)

{

Log.e("TTKJsonDecode", "No message\_id key in data");

return null;

}

TTKProtocol.MessageID msgid =

TTKProtocol.messageIDbyString(msgidString);

if (msgid == null)

{

Log.e("TTKJsonDecode", "Unknown messageId [" + msgidString + "]");

return null;

}

TTKMessage message = TTKMessage.createMessage(msgid);

if (message == null)

{

Log.e("TTKJsonDecode", "Cant create message for msgid [" +

msgidString + "]");

return null;

}

message.setTTKVersion(ttk\_ver);

message.setProtocol(this);

for (RequestFields f : message.getRequestFields())

{

String key = request\_tags.get(f);

TTKDecoder decoder = tag\_decoders.get(f);

if (decoder == null)

{

Log.e("TTKMessage", "parseMessage: unknown decoder for " +

f.toString());

return null;

}

Object val;

try

{

val = json.get(key);

}

catch (JSONException e)

{

continue;

}

if (val == null)

{

continue;

}

try

{

decoder.decode(message, val);

Log.d("TTKDecode", "Decoded " + f.toString() + " Value: " +

String.valueOf(val));

}

catch (Exception e)

{

throw new TTKFormatException(e);

}

}

return message;

}

private int getIntOrDefault(JSONObject json, String key, int d)

{

try

{

return json.getInt(kTTK\_VERSION\_FIELD);

}

catch (JSONException e)

{

return d;

}

}

private JSONObject parseJson(String s)

{

try

{

return new JSONObject(s);

}

catch (JSONException e)

{

e.printStackTrace();

return null;

}

}

protected static void setEncoder(TTKProtocol.ResponseFields f,

TTKSimpleEncoder encoder)

{

tag\_simple\_encoders.put(f, encoder);

}

protected static void setDecoder(TTKProtocol.RequestFields f, TTKDecoder

decoder)

{

tag\_decoders.put(f, decoder);

}

static Object encodeTransactionAmount(TTKTransactionMessage msg)

{

if (msg.getTransactionAmount() == null)

{

return null;

}

return Long.parseLong(msg.getTransactionAmount().getStrippedStr());

}

static Object encodeInvoiceNumber(TTKTransactionMessage msg)

{

return msg.getInvoiceNumber();

}

static Object encodeAuthorizationID(TTKTransactionMessage msg)

{

return msg.getAuthorizationID();

}

static Object encodeResponseCode(TTKTransactionMessage msg)

{

return msg.getResponseCode();

}

static Object encodePAN(TTKTransactionMessage msg)

{

return msg.getPAN();

}

static Object encodeDate(TTKTransactionMessage msg) throws

TTKFormatException

{

return TTKConvertors.formatedDateOrNull("ddMMyyyy", msg.getDateTime());

}

static Object encodeTime(TTKTransactionMessage msg) throws

TTKFormatException

{

return TTKConvertors.formatedDateOrNull("HHmmss", msg.getDateTime());

}

static Object encodeIssuerName(TTKTransactionMessage msg)

{

return msg.getIssuerName();

}

static Object encodeMerchantNo(TTKTransactionMessage msg)

{

return msg.getMerchantNo();

}

static Object encodeProcessingCode(TTKTransactionMessage msg)

{

return msg.getProcessingCode();

}

static Object encodeBatchNo(TTKTransactionMessage msg)

{

return msg.getBatchNo();

}

static Object encodeReceipt(TTKTransactionMessage msg)

{

return msg.getReceipt();

}

static Object encodeTerminalID(TTKTransactionMessage msg)

{

return msg.getTerminalID();

}

static Object encodeReceiptPDS(TTKTransactionMessage msg)

{

return msg.getReceiptPDS();

}

static Object encodeApprove(TTKTransactionMessage msg)

{

return msg.getApprove();

}

static Object encodeLocalTime(TTKTransactionMessage msg) throws

TTKFormatException

{

return TTKConvertors.formatedDateOrNull("yyyyMMddHHmmss",

msg.getLocalTime());

}

static void decodeTransactionAmount(TTKTransactionMessage msg, Object

data)

{

Amount a = new Amount();

if (data.getClass().equals(Long.class))

a.setStrippedStr(((Long) data).toString());

else

a.setStrippedStr(((Integer) data).toString());

msg.setTransactionAmount(a);

}

static void decodeTransactionMode(TTKTransactionMessage msg, Object data)

{

boolean no\_discount = getBoolOrDefault((JSONObject) data, "no\_discount",

false);

boolean search\_last = getBoolOrDefault((JSONObject) data, "search\_last",

false);

boolean no\_result\_indication = getBoolOrDefault((JSONObject) data,

"no\_result\_indication",false);

boolean no\_receipts = getBoolOrDefault((JSONObject) data, "no\_receipts",

false);

byte[] res = new byte[]{0x00};

res[0] = makeByte(no\_discount, search\_last, no\_result\_indication, no\_withdraw, pre\_input, ignore\_jrn\_status, no\_receipts, no\_name\_indication);

msg.setTransactionMode(res);

}

private static boolean getBoolOrDefault(JSONObject data, String key,

boolean b)

{

try

{

return data.getBoolean(key);

}

catch (JSONException e)

{

return b;

}

}

static void decodeInvoiceNumber(TTKTransactionMessage msg, Object data)

{

msg.setInvoiceNumber((Integer) data);

}

static void decodeAuthorizationID(TTKTransactionMessage msg, Object data)

{

msg.setAuthorizationID((String) data);

}

static void decodeCurrency(TTKTransactionMessage msg, Object data)

{

msg.setCurrency((Integer) data);

}

static void decodeInterfaceLanguage(TTKTransactionMessage msg, Object

data)

{

msg.setInterfaceLanguage((String) data);

}

static void decodeLocalTime(TTKTransactionMessage msg, Object data)

{

SimpleDateFormat format = new SimpleDateFormat("yyyyMMddHHmmss");

try

{

msg.setLocalTime(format.parse((String) data));

return;

}

catch (ParseException e)

{

e.printStackTrace();

}

msg.setLocalTime(null);

}

static void decodePAN(TTKTransactionMessage msg, Object data)

{

msg.setPAN((String) data);

}

static void decodeHostConfig(TTKTransactionMessage msg, Object data)

{

String login = getStringOrDefault((JSONObject) data, "login", null);

String password = getStringOrDefault((JSONObject) data, "password",

null);

HostConfiguration hc = new HostConfiguration();

hc.passwd = password;

hc.login = login;

msg.setHostConfig(hc);

}

}