Payment protocol PaymentsAPI v1.0.4

Document change history:

Version	**Date**	**Author**	**Comment**
1.0.0	02.03.2021	Menkov. V.	Document creation
1.0.1	15.03.2021	Menkov. V.	Added section “Certificate Conversion”
1.0.2	19.04.2021	Menkov. V.	Added description of Payment/Check method
1.0.3	19.04.2021	Menkov. V.	Added description of Balance method
1.0.4	15.10.2021	Menkov. V.	Added Pending status

General description of the protocol

Technology: REST HTTP API.

Data format: JSON.

List of methods:

- Init - payment initialization
- Confirm - confirmation of payment

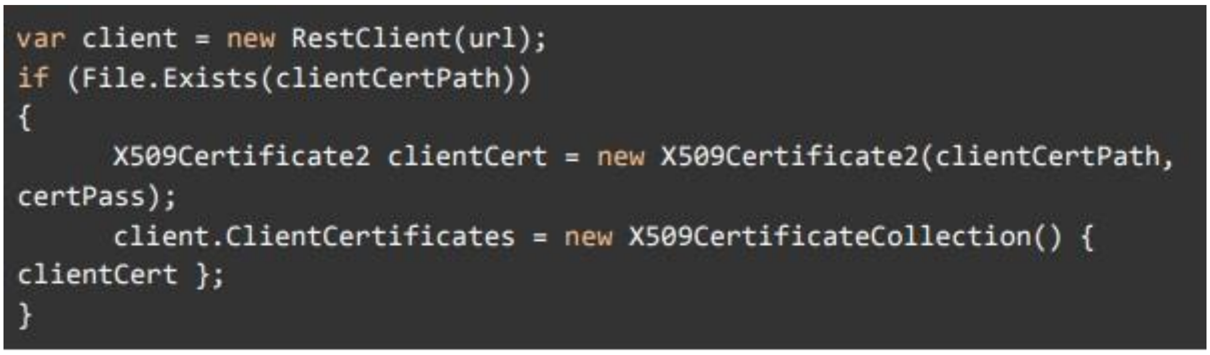
Authentication:

- Client certificate
- HMAC signature

Authentication

Client certificate

Each request must contain information about the client certificate. An example of using a client certificate in the code of a test utility:



```
var client = new RestClient(url);
if (File.Exists(clientCertPath))
{
      X509Certificate2 clientCert = new X509Certificate2(clientCertPath,
certPass);
      client.ClientCertificates = new X509CertificateCollection() {
clientCert };
}
```

where **clientCertPath** - path to certeficate, **certPass** - pswd of certeficate.

Signature HMAC

If

```
CLIENT = Client ID,
BODY = request body (JSON),
TS = unix timestamp (UTC+0) in milliseconds (calculated each time before
request), API-SECRET = client's private key,
```

then the message to sign would be: **MESSAGE = CLIENT + TS + BODY**.

We calculate the signature as HMAC-SHA256 from MESSAGE using a private key API-SECRET and then convert the byte array to string HEX - format:

SIGN = HEX (HMAC-SHA256(**MESSAGE**, **API-SECRET**)).

When sending a request, fill in the following HTTP headers:

```
RP-CLIENT = CLIENT
RP-TS = TS
RP-SIGN = SIGN
```

Пример:

```
CLIENT = N1Lin11

BODY = {
"clientTranId": "130",
"account": "282380",
"amount": 54.80,
"commissionAmount": 1.50,
"currency": "LYD",
"operatorCode": 5293
}

TS =1614696692368
 API-SECRET = 12345
```

then the message to be signed will be:

```
MESSAGE = N1Lin111614696692368
{
"clientTranId": "130",
"account": "282380",
"amount": 54.80,
"commissionAmount": 1.50,
"currency": "LYD",
"operatorCode": 5293
}
```

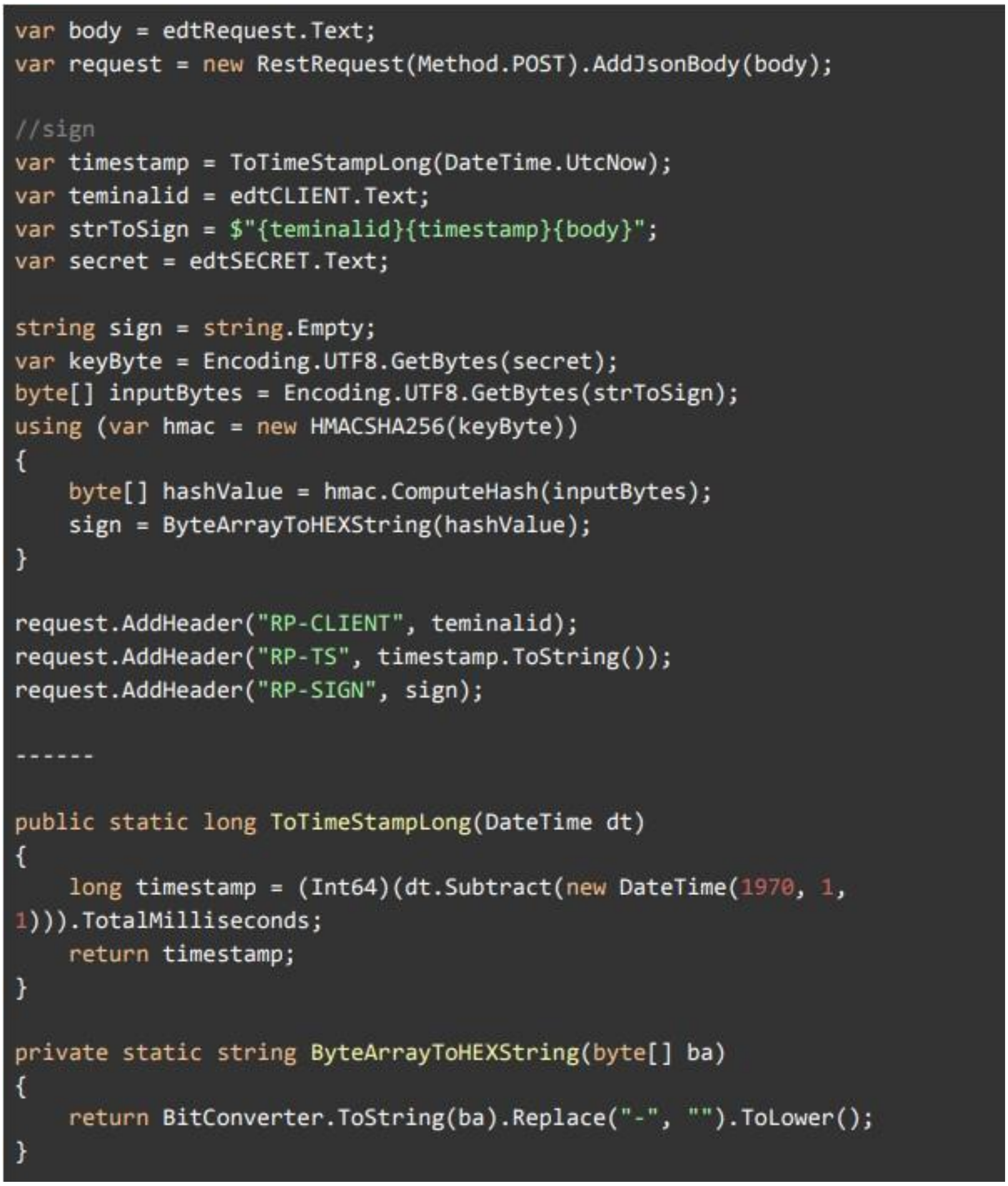
We calculate the signature as HMAC-SHA256 from **MESSAGE** using a private key **API-SECRET** and then convert the byte array to string HEX - format:

SIGN = HEX (HMAC-SHA256(**MESSAGE**, **API-SECRET**)) = 108b03d50319b9422df3a121991c474fa441df41f94c62683373099d473275b0.

When sending a request, fill in the following HTTP headers:

```
RP-CLIENT N1Lin11
RP-TS 1614696692368
RP-SIGN 108b03d50319b9422df3a121991c474fa441df41f94c62683373099d473275b0
```

Sample code from test utility:



```
var body = edtRequest.Text;
var request = new RestRequest(Method.POST).AddJsonBody(body);

//sign
var timestamp = ToTimeStampLong(DateTime.UtcNow);
var teminalid = edtCLIENT.Text;
var strToSign = $"{teminalid}{timestamp}{body}";
var secret = edtSECRET.Text;

string sign = string.Empty;
var keyByte = Encoding.UTF8.GetBytes(secret);
byte[] inputBytes = Encoding.UTF8.GetBytes(strToSign);
using (var hmac = new HMACSHA256(keyByte))
{
    byte[] hashValue = hmac.ComputeHash(inputBytes);
    sign = ByteArrayToHEXString(hashValue);
}

request.AddHeader("RP-CLIENT", teminalid);
request.AddHeader("RP-TS", timestamp.ToString());
request.AddHeader("RP-SIGN", sign);

------

public static long ToTimeStampLong(DateTime dt)
{
    long timestamp = (Int64)(dt.Subtract(new DateTime(1970, 1,
1))).TotalMilliseconds;
    return timestamp;
}

private static string ByteArrayToHEXString(byte[] ba)
{
    return BitConverter.ToString(ba).Replace("-", "").ToLower();
}
```

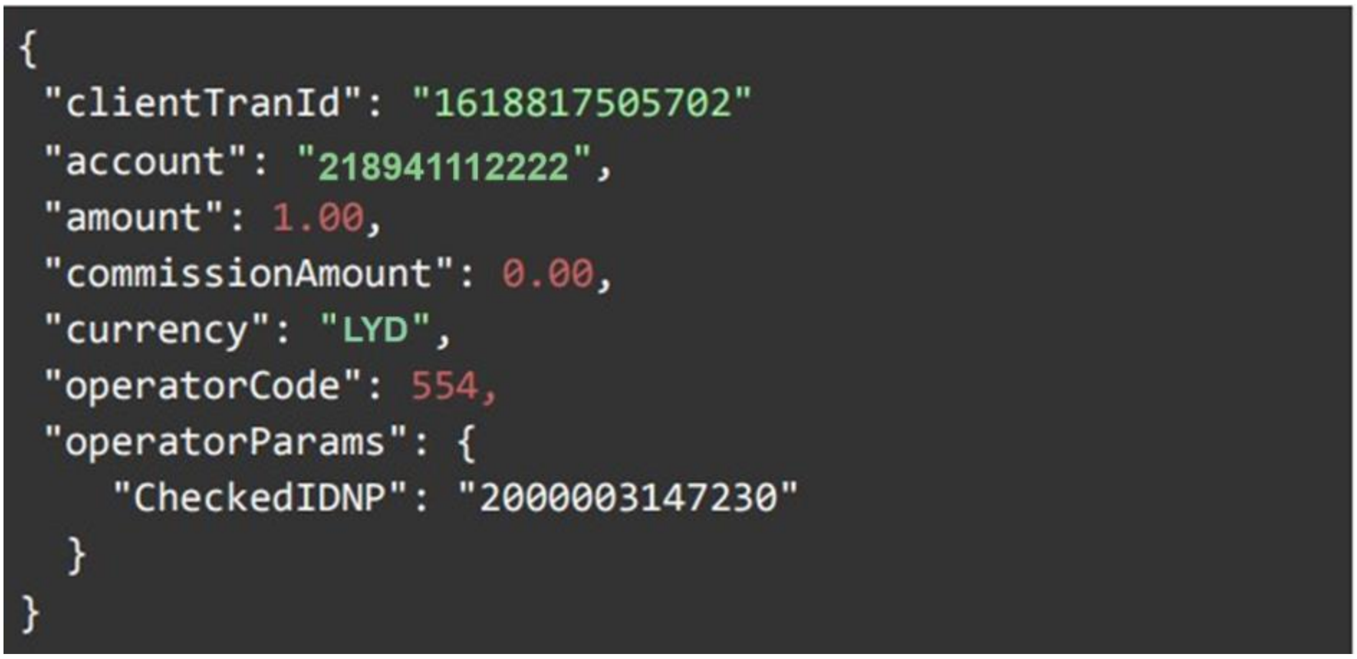
Service methods

[Payment/Init] Payment initialization method

Request type: POST.

This is the first request in a billing session. At this stage, various checks are carried out for the correctness of the payment data and for the possibility of performing this operation.

Request example:

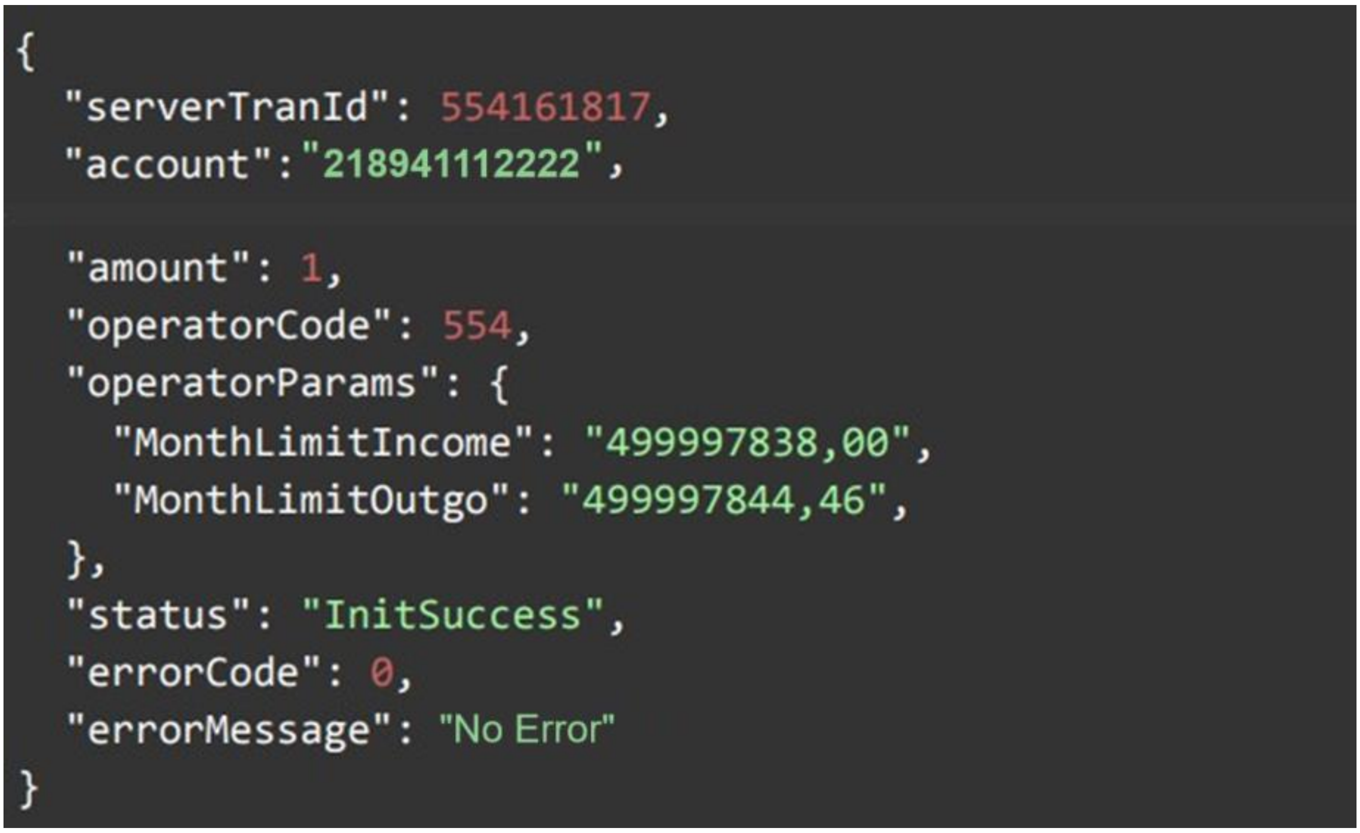


```
{
 "clientTranId": "1618817505702"
 "account": "218941112222",
 "amount": 1.00,
 "commissionAmount": 0.00,
 "currency": "LYD",
 "operatorCode": 554,
 "operatorParams": {
    "CheckedIDNP": "2000003147230"
  }
}
```

where

```
clientTranId - transaction number in the client's system (optional),
account - account, amount - amount to pay, commissionAmount - commission
amount, currency - currency, operatorCode - operator code,
operatorParams - incoming operator parameters (each operator has its own
parameters).
```

Sample response:



```
{
  "serverTranId": 554161817,
  "account":"218941112222",

  "amount": 1,
  "operatorCode": 554,
  "operatorParams": {
    "MonthLimitIncome": "499997838,00",
    "MonthLimitOutgo": "499997844,46",
  },
  "status": "InitSuccess",
  "errorCode": 0,
  "errorMessage": "No Error"
}
```

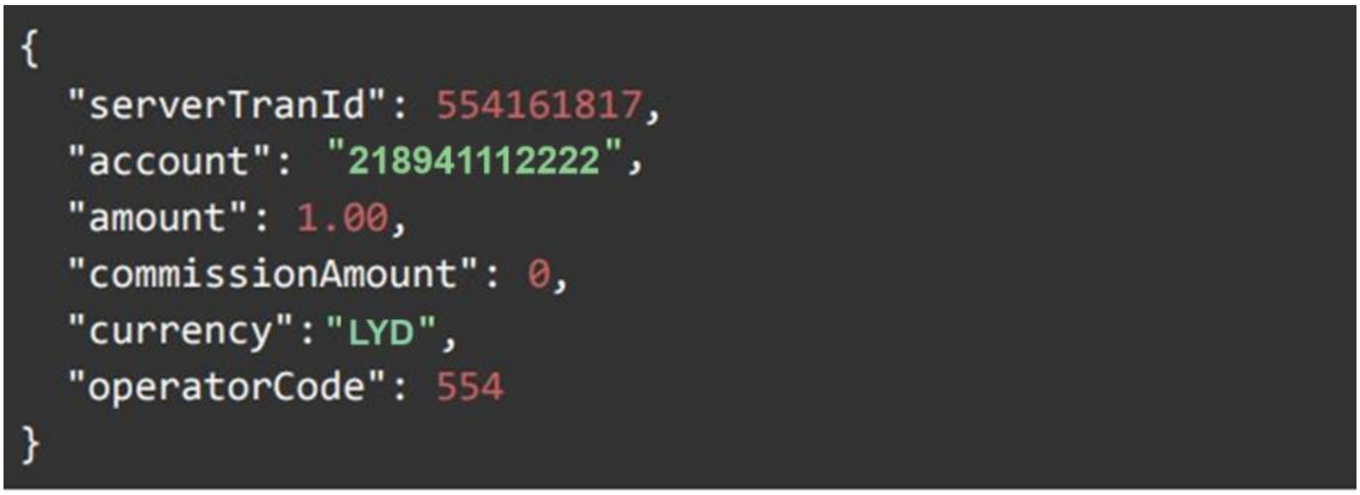
where

```
serverTranId - transaction identifier (This value then needs to be
substituted in the Confirm request), operatorParams - various operator
parameters status - transaction status (see description of statuses)
errorCode - error code (see description of error codes) errorMessage -
error description operatorParams - outgoing operator parameters
```

[Payment/Confirm] Payment confirmation method

Request type: POST. This method is called after successful initialization to process the payment.

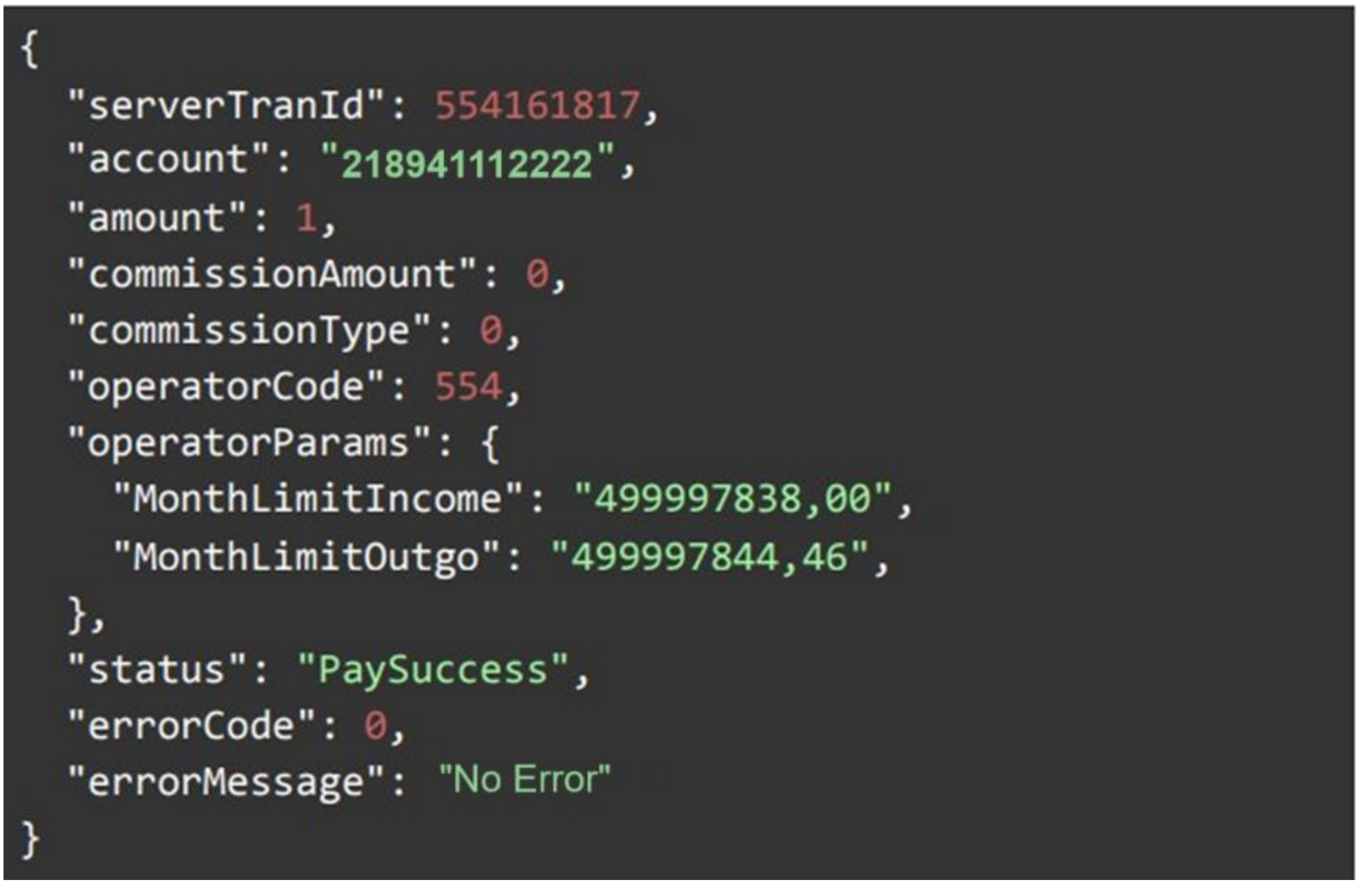
Example request:



```
{
  "serverTranId": 554161817,
  "account": "218941112222",
  "amount": 1.00,
  "commissionAmount": 0,
  "currency":"LYD",
  "operatorCode": 554
}
```

where **serverTranId** - transaction number (comes in response to /Init).

Sample answer:

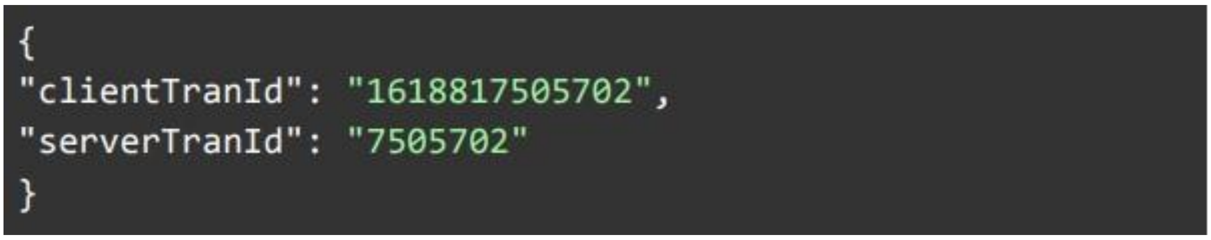


```
{
  "serverTranId": 554161817,
  "account": "218941112222",
  "amount": 1,
  "commissionAmount": 0,
  "commissionType": 0,
  "operatorCode": 554,
  "operatorParams": {
    "MonthLimitIncome": "499997838,00",
    "MonthLimitOutgo": "499997844,46",
  },
  "status": "PaySuccess",
  "errorCode": 0,
  "errorMessage": "No Error"
}
```

[Payment/Check] Payment verification method

Request type: POST. This method is called to clarify the status of the sent payment or check if the operation ended with an error or the connection was interrupted.

Request example:



```
{
"clientTranId": "1618817505702",
"serverTranId": "7505702"
}
```

where

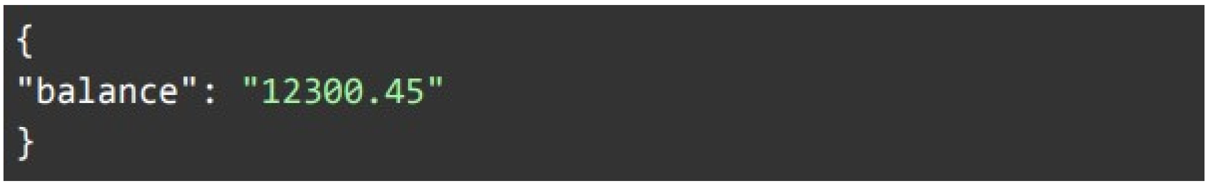
```
clientTranId - transaction number in the client's system serverTranId -
transaction number in the RunPay system (comes in response to Init/Confirm)
```

It is enough to fill in one of the parameters. If both parameters are filled, then the priority serverTrandId. The response is identical to the response to the Init/Confirm methods.

[/Balance] Payment verification method

Request type: GET. This method is used to query the user's balance.

Answer example:



```
{
"balance": "12300.45"
}
```

Basic scenario for making a payment

- Sending request /Init
- From the answer we take serverTranId and substitute in the request /Confirm We send
- request /Confirm

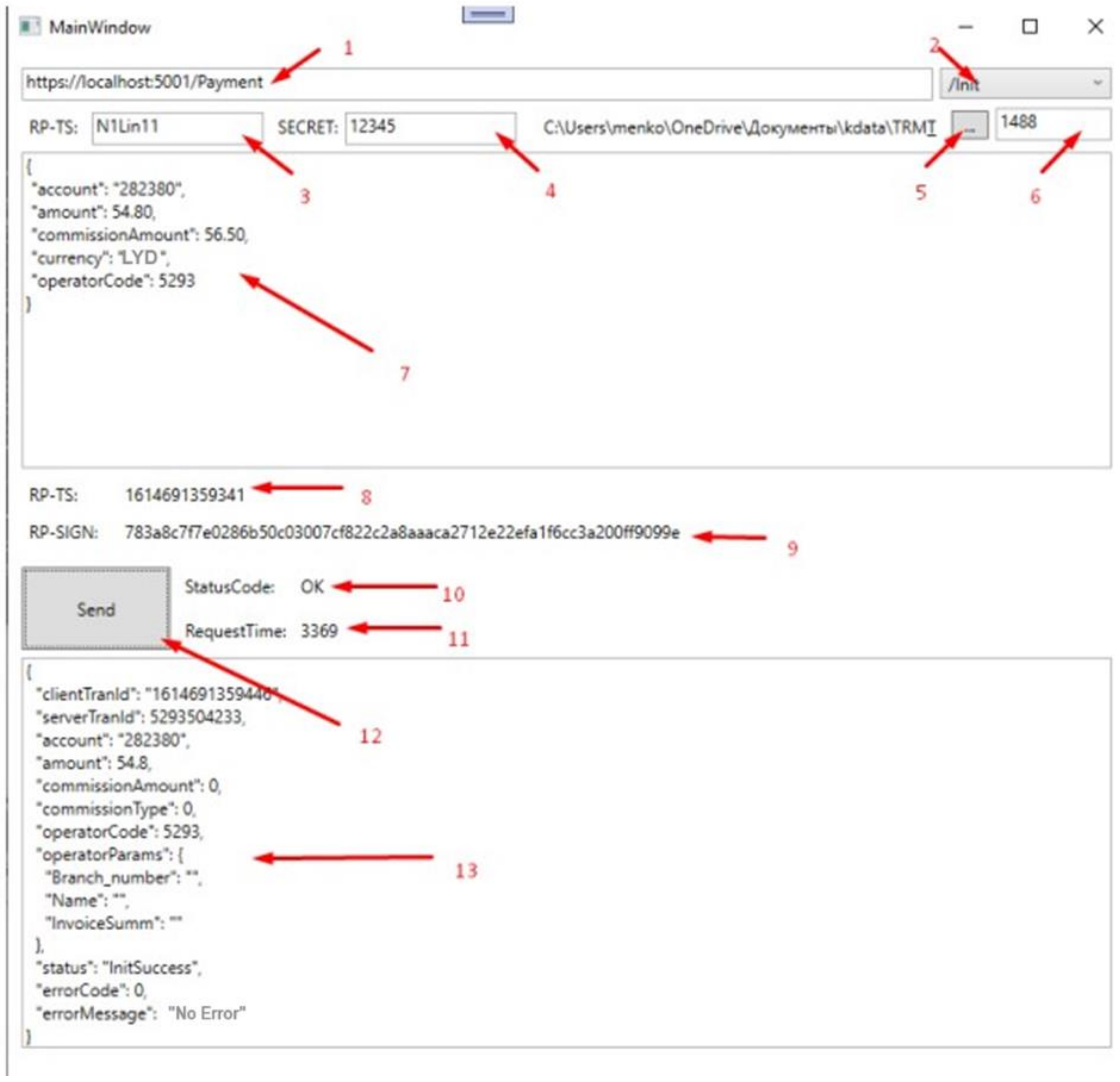
Converting the certificate

Some cases instead of pfx the certificate needs a different format such as a pem / key pair. For such purposes, you need to use the public utility open_ssl, below are examples of the required commands:

1. Making a root certificate: openssl pkcs12 -in N4XBB.pfx -cacerts -nokeys -out ca.pem
2. Making a pem with a client certificate: openssl pkcs12 -in N4XBB.pfx -clcerts -nokeys -out cert.pem
3. Making a pem with a private key: openssl pkcs12 -in N4XBB.pfx -nocerts -out key.pem

Test utility

The TestClient utility has been created to test the API. You can use it to send with a signature.



https://localhost:5001/Payment /Init

RP-TS: N1Lin11 SECRET: 12345 C:\Users\menko\OneDrive\Документы\kdata\TRMI ... 1488

```
{
 "account": "282380",
 "amount": 54.80,
 "commissionAmount": 56.50,
 "currency": "LYD",
 "operatorCode": 5293
}
```

RP-TS: 1614691359341

RP-SIGN: 783a8c7f7e0286b50c03007cf822c2a8aaaca2712e22efa1f6cc3a200ff9099e

StatusCode: OK

RequestTime: 3369

```
{
 "clientTranId": "1614691359440",
 "serverTranId": 5293504233,
 "account": "282380",
 "amount": 54.8,
 "commissionAmount": 0,
 "commissionType": 0,
 "operatorCode": 5293,
 "operatorParams": {
  "Branch_number": "",
  "Name": "",
  "InvoiceSumm": ""
 },
 "status": "InitSuccess",
 "errorCode": 0,
 "errorMessage":  "No Error"
}
```

1. API base address
2. Method
3. Client code (issued upon registration)
4. Client password (issued during registration)
5. Client certificate (issued upon registration)
6. Password from the certificate (issued during registration)
7. Request body
8. The generated timestamp value
9. The generated signature value
10. Query result
11. Lead time
12. Send request button

13. Response body

List of transaction statuses

- InitFail - payment initialization error
- InitPorcess - transaction validation is in progress
- InitSuccess - successful initialization of the payment
- PayFail - payment confirmation error
- PayProcess - the payment is being processed
- PaySuccess - payment completed successfully
- PayPending - deferred payment processing

List of error codes

0	No error
1	Incorrect query parameters
2	Request log not found
3	No active store or active business found
4	Repeat Request ID
5	No matching validation request found. Perhaps the wrong status of the transaction, ie. repeat command
6	Subagent point code is not registered in the database
7	Certificate expired according to DB
8	Phone number/account length is too long
9	No gateway found for online command
10	Operator blocked or subagent banned
11	Transaction date too old
12	Error adding payment to the queue
14	Certificate verification error
15	Invalid optional parameter format or missing required parameter
100	Data not found
101	Internal Server Error
34	Prohibited payment currency
114	Exceeding the limit for the period