Quick Start

This document contains instructions for setting up a Utopia p2p host in order to automate certain actions: making and receiving payments, and ordering goods and services by email or via personal messages. The consumers of such services may include, e.g., a cryptocurrency exchange, an exchange service, an advertising agency, a support bot, and other services.

To launch the Utopia p2p network client with API support, you need to complete a number of actions:

– Install the application

– Set up and enable API mode support

– Verify that settings are correct

All necessary operations are described in detail below.

Installation

The installation process varies depending on the OS.

Windows

To run the application, you need Windows 7, 8 or 10.

We recommend using the 64-bit version of the OS.

Installation distribution is available here:

```
https://update.u.is/downloads/windows/latest/UtopiaX64.exe
```

The installation process is very easy - all you need to do is follow the steps of the Install Wizard. We recommend that you don't run the application at the last stage (uncheck the application launch box).

MacOS

To run the application, you need to use the operating system version 10.12 or later.

Installation distribution is available here:

```
https://update.u.is/downloads/macos/utopia-latest.dmg
```

The dmg file installation process is standard for MacOS.

Linux

Only 64-bit OSs are supported. The bit depth of the OS can be verified using the following command:

```
uname -r
```

If the response to this command is xxxxx.x86_64, then this a 64-bit OS.

The list of supported OSs, RPM-like:

OS Name	Verification Command	Sample OS Response
RHEL	cat /etc/redhat-release or cat /etc/issue	Red Hat Enterprise Linux Server release 7.6 (Maipo)
CentOS	cat /etc/centos-release or cat /etc/issue	CentOS Linux release 7.6.1810 (Core)
Fedora	cat /etc/fedora-release or cat /etc/issue	Fedora release 10 (Cambridge)
SUSE	cat /etc/SuSE-release or cat /etc/issue	SUSE Linux Enterprise Server 12 (x86_64)

DEB-like OSs:

OS Name	Verification Command	Sample OS Response
Ubuntu	cat /etc/debian_version or cat /etc/issue	9.5 or Debian GNU/Linux 9
Debian	cat /etc/lsb-release or cat /etc/issue	DISTRIB_ID=Ubuntu

To run the application, you need to:

1. Install the following group of additional packages:

rpm-like distributions:

```
yum install -y libxkbcommon-x11 mesa-libGL pulseaudio-libs-glib2 fontconfig
```

+ all the dependencies that will be offered during the installation process

deb-like distributions:

```
apt-get install libx11-xcb1 libgl1-mesa-glx libpulse-mainloop-glib0
libfontconfig
```

+ all the dependencies that will be offered during the installation process

2. Download the latest Utopia client version

rpm-like distributions:

```
wget https://update.u.is/downloads/linux/utopia-latest.x86_64.rpm
```

deb-like distributions:

```
wget https://update.u.is/downloads/linux/utopia-latest.amd64.deb
```

3. Install the client using the command:

```
rpm -ivh utopia-1.0-XXXX.x86_64.rpm
```

or

```
dpkg -i utopia-1.0-XXXX.amd64.deb
```

Setting Up and Enabling API Mode Support

Depending on the configuration used and the display availability on your machine, as well as depending on personal preferences, the application can be launched in the graphical interface mode or in the headless mode.

Important: Your Utopia p2p host must stay on all the time when working with the API. To ensure stable operation of the service, running multiple applications from the same IP address is not allowed.

We strongly recommend that you keep your Utopia p2p host on all the time.

Using Graphical Interface (GUI)

1. Launch the application

2. Create a new cryptocontainer

Follow the on-screen wizard instructions to create a new cryptocontainer.

```
How to create an account in Utopia Network?

Run Utopia application.
Click "Create new account" button
At the "Create Your Utopia Account" page enter your Nickname. Optionally,
you may enter your first and last names. Please note that your Nickname and
first/last names (if entered) will be visible to your authorized contacts.
Click "Next"
Leave the default path, but take note of it. This is the path to your
Encrypted Container that will be created on your computer. The purpose of
the Encrypted Container is to store your Utopia data, such as your private
key, uMails, files, uWallet, chat history, contacts and transactions history
in encrypted form. You may select any folder on your computer if you wish.
Enter the password to your Encrypted Container twice. Make sure that you
have chosen a strong password. Be sure to remember your password and never
store it in plain text at your computer. Lost passwords cannot be recovered,
resulting in the permanent lost of access to your Utopia account.
Important: Make sure that you backup your Encrypted Container regularly and
store your password somewhere safe, as loss of the container or password will
result in the permanent loss of your Utopia account data and account access.
Click "Next" button to proceed to the next step.
Familiarize yourself with information on mining within the Utopia ecosystem.
By default mining is enabled, however you may disable it by unchecking the
"Enable Mining" checkbox. Click "Finish".
Your new Utopia account has been created.
```

Specify the location for saving the cryptocontainer and enter a strong password. Typical cryptocontainer storage location:

On Windows, the file is located in:

```
C:\Users\%username%\AppData\Roaming\Utopia\Utopia Client\db
```

Linux:

```
home\%username%\.local\share\Utopia\Utopia Client\db
```

MacOS:

```
home\%username%\Library\Application Support\Utopia\Utopia Client\db
```

3. Configure working with the API

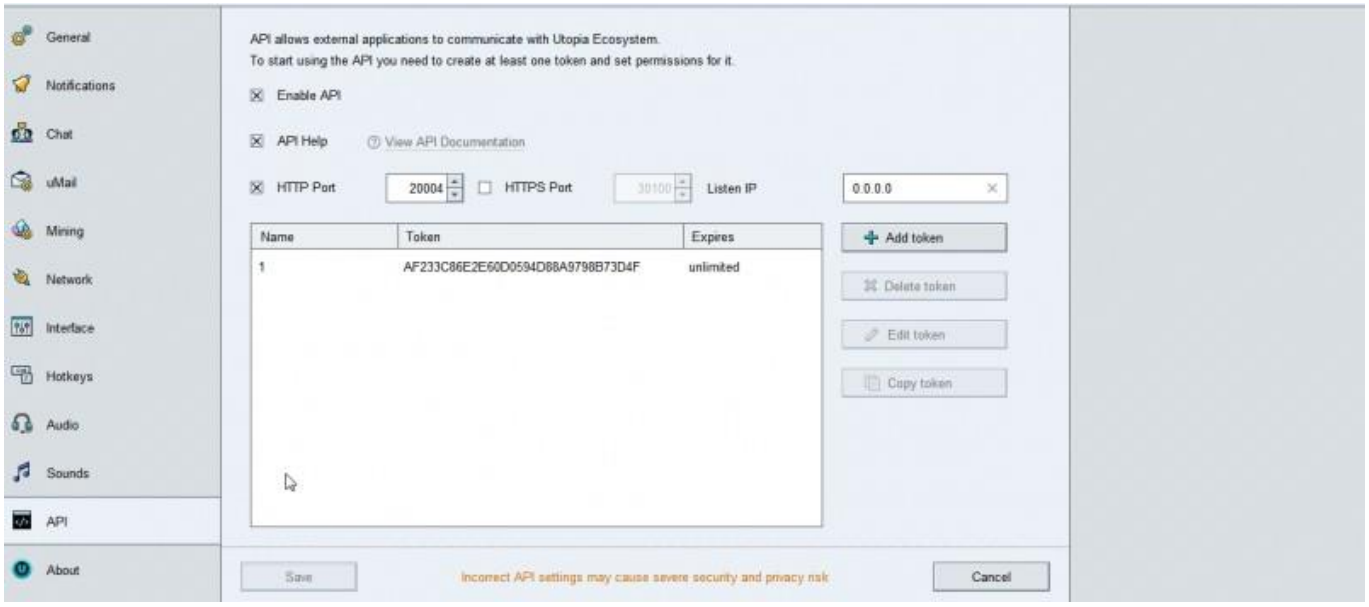
Go to Tools→Settings→API.

Check the Enable API box.

Depending on your preferences, enable communication via the http and/or https protocol and set the desired port.

To increase the connection security, you can specify the address from which the connection to your Utopia host will be established in order to run automated scripts. In this case, set the allowed address in the Listen IP field; otherwise leave 0.0.0.0 to allow connections from any IP address.

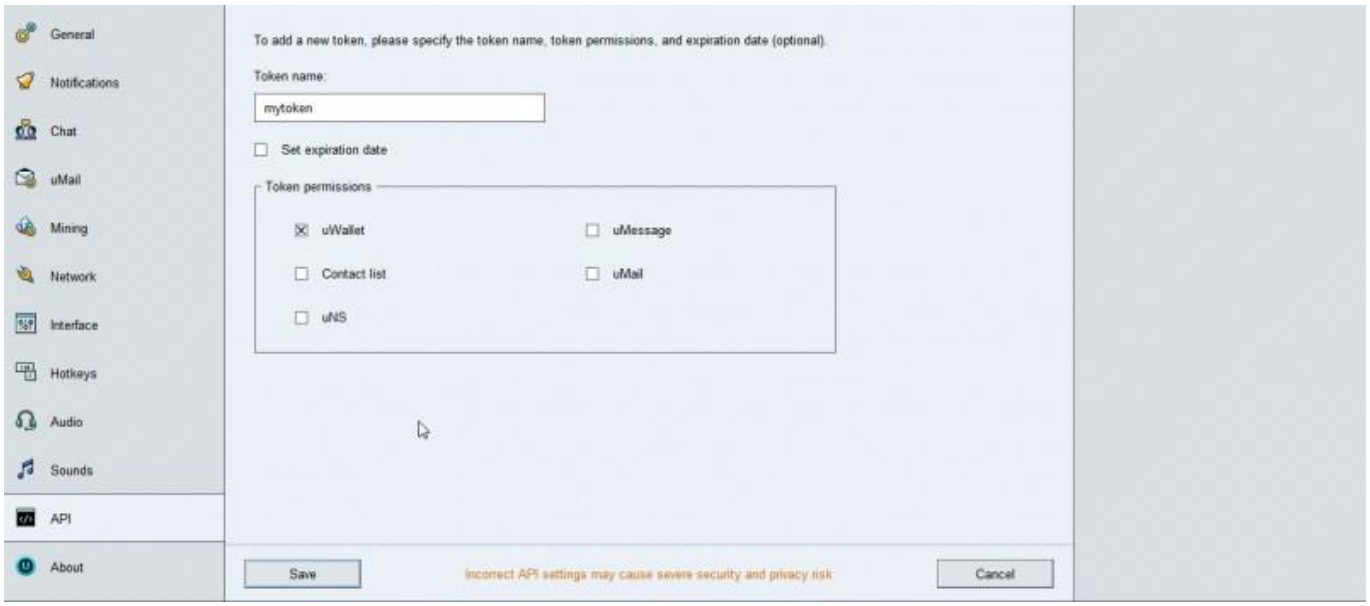
An example of the correct settings is shown in the screenshot below.



The last step is to get a token to be able to execute API requests.

Click the Add token button and then specify your token name, scope, and lifetime, if necessary.

An example of the correct settings is shown in the screenshot below.



Headless Mode

This mode enables launching (for example, via ssh) the client p2p functionality in a non-display configuration with API support.

You can use the keys to launch the application with the base, password and headless mode indicated. Example:

```
cd /opt/utopia/messenger/Utopia && ./utopia --db /root/merchant.db --pwd
merchant_password --headless
```

where the keys are:
`--db` indicating the path to the cryptocontainer
`--pwd` indicating the cryptocontainer password

If the base is not created, you can create it using the following command:

```
cd /opt/utopia/messenger/Utopia && ./utopia --headless --create --
db=/path/to/db/sample.db --pwd 123
```

For convenience, the application can be launched using a configuration file, e.g.:

```
# Database password
userPassword=123

# Database path
userDatabase=/path/to/db

# List of API tokens separated by commas
apiUserTokens=AB3FF5AFCCC85347F43AB6350193D87E

# Enable API
apiEnabled=true
apiHTTPEnabled=true
apiHTTPSEnabled=true

# API HTTP port
apiPort=10010

# API HTTPS port
apiSslPort=10100

# API interactive help enabled (/api/help)
apiHelpEnabled=true

# Path to SSL certificate for HTTPS API
# apiCertificatePath=/path/to/cert.pem
# apiPrivateKeyPath=/path/to/privatekey.pem

# API listen address
#apiListenAddress=0.0.0.0
```

The **apiUserTokens** key allows you to explicitly create a token without any restrictions for completing all types of API requests. Use the HEX string format that is 32 characters long.

Launching the application using the configuration file:

```
cd /opt/utopia/messenger/Utopia && ./utopia --headless --
configPath=/path/to/config.cfg
```

You should create and save a backup copy of your cryptocontainer regardless of how you use the application. You can do this by simply copying the .db file.

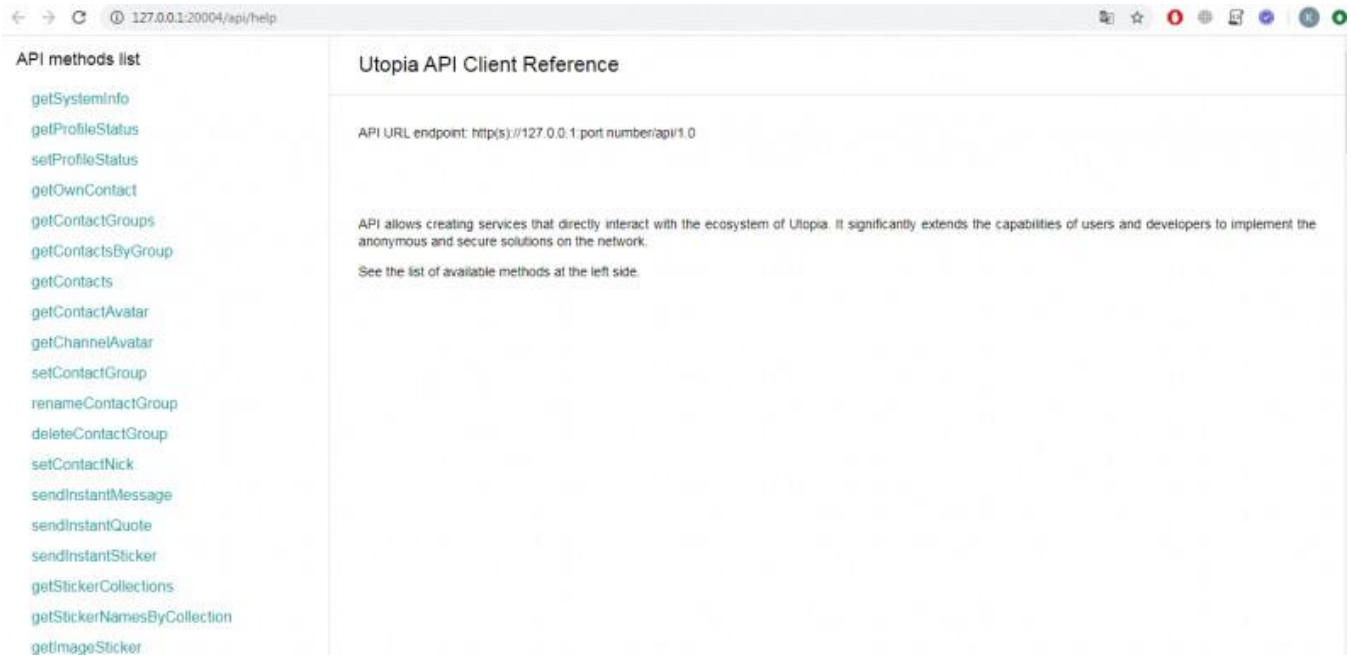
Verifying Correct Settings

If you are using the application with a graphical interface, go to Tools→Settings→API.

Enable API Help mode.

Click on the link to the right of the checkbox.

A window will open in the browser selected in your system by default. You can use it to get information about all the available methods and the format of requests and returned responses. Use the created non-expired token to ensure correct operation.



If using headless mode, specify the following in the configuration file or the application launch key:

```
apiHelpEnabled=true
```

A page with information on the available methods can be accessed via http or https with the specified port at:

```
%protocol%://127.0.0.1:%port%/api/help
e.g.: http://127.0.0.1:20004/api/help
```

Another option for checking the functionality of the specified settings is to send a POST request to:

```
%protocol%://127.0.0.1:%port%/api/1.0
e.g.: http://127.0.0.1:20004/api/1.0
```

Sample request

```
{
    "method": "getSystemInfo",
    "token": "3A2E92F178084F5ACEFDFE209DC35792"
}
```

Sample response

```
{
    "result": {
        "buildAbi": "x86_64-little_endian-llp64",
        "buildCpuArchitecture": "x86_64",
        "build_number": "1.0.5958",
        "currentCpuArchitecture": "x86_64",
        "netCoreRate": 25,
        "networkCores": 4,
        "networkEnabled": true,
        "numberOfConnections": 4,
        "packetCacheSize": 52740,
        "uptime": "04:07:58"
    },
    "resultExtraInfo": {
        "elapsed": "0"
    }
}
```

Congratulations! Setup mode completed successfully.

Examples of Typical User Scenarios

User library description and usage examples

A description of the Utopia Client API user library along with the examples of using the implemented calls with Python can be found below.

Initializing a Utopia class object

APIURL - Socket for working with API

PASSWORD - Token used

interval - Delay between consecutive executions of requests

```
import api
u=api.Utopia(APIURL, PASSWORD, interval=1)
```

General information on called methods

Calling a Utopia class method returns the status of the completed request and the request response data structure.

```
status, data = u.someMethod()
if status:
    #some data processing
    print(data)
```

Most methods allow implicitly obtaining the following arguments:

`sortBy` - indicates the need to sort the output Format used: `propertyName1:sortingorder,propertyName2:sortingorder` `offset` - offsets the output, prunes the specified `N` of initial results `limit` - limits the output

Example of setting and resetting filters:

```
u.genericFilter("name", 5, 100)

u.genericFilterClear()
```

Sample Codes

Finance

Getting system information

Displaying the version number of the application used

```
status, data = u.getSystemInfo()
if status:
    print(data["build_number"])
```

Getting data on the current account balance

```
status, data = u.getBalance()
if status:
    print("You balance: " + str(data))
```

Getting information on existing cards

```
status, cards = u.getCards()
if status:
```

```
    for card in cards :
        print("Card name is: " + card["name"] + "\n")
```

Transferring cryptons

```
if u.sendPayment("AnyPublicKey", 3.141592653):
    print("Success")
```

Creating a voucher

```
if u.createVoucher(3.141592653, "MyCardId"):
    print("Success")
```

Using a voucher with funds credited to the specified card

```
if u.useVoucher("VoucherId", "MyCardId"):
    print("Success")
```

Deleting a voucher with funds credited to the specified card

```
if u.deleteVoucher("VoucherId", "MyCardId"):
    print("Success")
```

Getting information on vouchers issued

```
status, vouchers= u.getVouchers()
if status:
    print("Total "+ str(len(vouchers))+" vouchers codes: ")
    for voucher in vouchers:
        print(voucher ["voucherid"]+"\n")
```

Getting invoice information

```
status, invoices= u.getInvoices()
if status:
    print("Total "+ str(len(invoices))+" invoices: ")
    for invoice in invoices:
        print("Id: " + invoice ["invoiceid"] + " amount: " + str(invoice
["amount"]) + "\n")
```

Sending an invoice

```
if u.sendInvoice("AnyCardId", 1.00, "Some comment"):
    print("Success")
```

Paying an invoice

```
if u.acceptInvoice("AnyInvoiceId"):
    print("Success")
```

Rejecting an invoice

```
if u.declineInvoice("AnyInvoiceId"):
    print("Success")
```

Canceling an invoice

```
if u.cancelInvoice("AnyInvoiceId"):
    print("Success")
```

Get a portion of financial transaction history

```
status, transactions = u.getFinanceHistory("ALL_VOUCHERS", "0", ""):
if status:
    for transaction in transactions:
        print("Transaction type: " + transaction["name"] + "\n")
```

Add card

```
if u.addCard("#77f442", "name", "a1b2"):
    print("Success\n")
```

Delete card

```
if u.deleteCard("A0950072D9261480"):
    print("Success\n")
```

Get information on the financial system

```
status, data = u.getFinanceSystemInformation()
if status:
    print(data)
```

Communications

Getting a list of contacts

```
status, contacts= u.getContacts("uto") # any nick like uto_pia
if status:
    print("Total "+ str(len(contacts))+" contacts: ")
    for contact in contacts:
        print("Nick: " + contact ["nick"] + " is your friend: " +str(contact
["isFriend"]) + "\n")
```

Send authorization request

```
if u.sendAuthorizationRequest("PublicKey"):
    print("Success\n")
```

Accept authorization request

```
if u.acceptAuthorizationRequest("PublicKey"):
    print("Success\n")
```

Cancel authorization request

```
if u.rejectAuthorizationRequest("PublicKey"):
    print("Success\n")
```

Delete contact

```
if u.deleteContact("PublicKeyOrNick"):
    print("Success\n")
```

Change contact group assignment

```
if u.setContactGroup("PublicKey", "GroupName"):
    print("Success\n")
```

Change contact nickname

```
if u.setContactNick("PublicKey", "NewNick"):
    print("Success\n")
```

Sending a chat message

```
if u.sendInstantMessage("AnyNick", "Hi"):
    print("Success\n")
```

Get message list

```
status, messages = u.getContactMessages("PK", limit, offset):
if status:
    for message in messages:
        print(message)
```

Get email list

```
status, mails = u.getEmailFolder(FolderType, ""):
if status:
    for mail in mails:
        print("mail id:"+str(mail) )
```

Get email information

```
status, mail = u.getEmailById("SomeEmailId"):
if status:
    print(mail["subject"])
```

Forward email

```
if u.sendForwardEmailMessage(Emailid, "Hello!", "AnyUserNick"):
    print("Success\n")
```

Reply by email

```
if u.sendReplyEmailMessage(Emailid, "Hello!", "AnyUserNick"):
    print("Success\n")
```

Delete email

```
if u.deleteEmail("SomeEmailId"):
    print("Success\n")
```

Sending emails to multiple recipients

```
if u.sendEmailMessage(["AnyNick","AnotherNick"], "e-mail subject", "Hello
from API =)"):
    print("Success\n")
```

Joining a channel

```
if u.joinChannel("ChannelId", "ChannelPassword"):
    print("Success\n")
```

Sending a channel message

```
if u.sendChannelMessage("ChannelId"):
    print("Success\n")
```

Leaving a channel

```
if u.leaveChannel("ChannelId"):
    print("Success\n")
```

Notifications

Notifications for all types of events (finances, IM, emails, channel messages and others) are received using the WebSocket mechanism.

The process of enabling and setting the priority port for listening is as follows:

```
status, port=u.setWebSocketState("true", 12348)
```

The result is the returned port selection status and, if the ordered port is already taken, the port to which it was possible to bind.

Sample code for listening to notifications:

```
websocket.enableTrace(True)
ws =
websocket.WebSocketapplication("ws://127.0.0.1:"+str(port)+"/UtopiaWSS?token
="
+ token, on_message = on_message, on_error = on_error, on_close = on_close,
on_open = on_open)
ws.on_open = on_open
```

on_message on_error on_close and on_open functions must be defined by the user.

Working with History and Force Update of Operations History

Enable auto update mining block history *

```
if u.enableHistoryMining(1)
    print("Success\n")
```

Get sync operation status

```
status = u.statusHistoryMining()
```

Possible operation statuses

```
0 = STATE_EMPTY
1 = STATE_IN_PROGRESS
2 = STATE_RECEIVED_RESPONSE
```

Get information on mining blocks

```
status, blocks = u.getMiningBlocks()
```

Get a portion of financial transaction history

```
status, transactions = u.getFinanceHistory("ALL_VOUCHERS", "", "", "", "",
0.0001, 100):
if status:
    for transaction in transactions:
        print("Transaction type: " + transaction ["name"] + "\n")
```

List of available filters:

```
ALL_CARDS
INCOMING_CARDS
```

```
OUTGOING_CARDS
CREATED_CARDS
DELETED_CARDS
ALL_TRANSFERS
INCOMING_TRANSFERS
OUTGOING_TRANSFERS
ALL_REQUESTS
AWAITING_REQUESTS
AUTHORIZED_REQUESTS
DECLINED_REQUESTS
CANCELED_REQUESTS
EXPIRED_REQUESTS
ALL_APPROVED_REQUESTS
CREATED_VOUCHERS
CREATED_VOUCHERS_BATCH
ACTIVATED_VOUCHERS
DELETED_VOUCHERS
ALL_VOUCHERS
ALL_MINING
ALL_POS
ALL_FEE
ALL_UNS_RECORDS
UNS_UNS_REGISTRATION
UNS_UNS_CHANGED
UNS_UNS_TRANSFERRED
UNS_UNS_DELETED
ALL_TRANSACTIONS

Filters can be combined with a comma: "ALL_CARDS,ALL_FEE"
```

Usecases

The most popular are operations of depositing or withdrawing funds. Let’s look at them in more detail.

Deposit

Minimal

Used when funds are not deposited frequently to the main account and/or the deposit amount is known in advance.

```
status, balance = u.getBalance()
print(balance)

# wait some operations
```

```
status, newbalance = u.getBalance()
print(newbalance)

if (newbalance > balance):
    print("Deposit\n")
```

Normal

Used in the usual mode when the payment can be identified by pk or by adding a special comment to the payment. In this case, you can get a history of recent operations. In the example below, transactions sorted by creation date are filtered and only the last 1 is displayed as a result.

The financial history is used with the incoming transactions filter. Filters can be combined. The analysis is based on the PK address of the funds' sender or the content of the comment regarding transaction ID.

```
u.genericFilter("created:desc", 0, 1)
status, transactionHistory = u.getFinanceHistory("INCOMING_TRANSFERS",
"","","","","","")

# check value of transactionHistory[0]["source_pk"]
# or check value of transactionHistory[0]["comments"] for some transaction
id
```

Maximum

The mode involves the maximum division of funds accounting.

An invoice is issued for a specific payer using their card. Payment is tracked using the transaction ReferenceNumber.

```
status, refnumber = u.sendInvoice ("id", "B7EB00274D1585F7", 0.5)

print(refnumber)

# wait for payment of the invoice

status, invoiceinfo = u.getInvoiceByReferenceNumber(refnumber)
print(invoiceinfo)
if (invoiceinfo["status"] == "Authorized"):
    print(invoiceinfo["amount"])
```

The division of the above examples into minimal-normal-maximum is conditional. You can combine the proposed options or use your own.

Withdrawal

In the withdrawal mode, you need to transfer funds and then make sure that the transaction was accepted by the network.

Minimal

The simple mode involves transferring funds and checking for the transfer confirmation by the network.

```
status, paymentid =
u.sendPayment("77D1832EBB35F233A1BB4B4AD9ECC7774F8F2000733EAA81F615A9173A487
A52", "id", "", 0.8)
print(paymentid)

# wait for network confirmation

u.genericFilter("created:desc", 0, 1)
status, transactionHistory = u.getFinanceHistory("OUTGOING_TRANSFERS",
"","","","","","")
print(transactionHistory)
```

Normal

The mode involves transferring funds by card number or the recipient's PK, possibly with the indication of transaction ID in the comments, and the subsequent verification of transfer confirmation by the network.

```
status, paymentid = u.sendPayment("", "id", "cardid", 0.8)
print(paymentid)

# wait for network confirmation
status, transactionHistory =
u.getFinanceHistory("", paymentid,"","","","","")
print(transactionHistory)
```

Maximum

The mode involves working with cards tied to payers' accounts. The invoice is received from the payee and the data is analyzed (for example, by the operation ID in the comment or the PK that requested the invoice, by date and other information). If everything is correct, the invoice is paid and the payment status is checked.

```
u.genericFilter("created:desc", 0, 1)
```

```
status, invoices = u.getInvoices("299C0011032F258C","",
"","","Awaiting","","","")

invoiceid=invoices[0]["invoiceid"]
status, refid= u.acceptInvoice(invoiceid)

# wait for network confirmation
status, transactionHistory =
u.getFinanceHistory("", paymentid,"","","","","")
print(transactionHistory)
```

The proposed scenarios are not the only ones that are true. For example, you can choose to issue a voucher and transfer its code to the funds' recipient. Or you can combine the proposed approaches as you like.

List of Available API Libraries

To execute API requests, you can use the raw format for transmitting json data using the POST method, or use the available official libraries:

- **API python**
- **API perl**
- **API asp classic**
- **API php**
- **API curl**
- **API ruby**