Getting Started

Overview

PyBand is a library written in Python used for interacting with BandChain. The library provides classes, methods, andprotobuf classes for the ease of sending transactions, querying data, OBI encoding, and wallet management.

The library is implemented based on gRPC-web protocol which sends HTTP/1.5 or HTTP/2 requests to a gRPC proxy server, before serving them as HTTP/2 to gRPC server.

This library is only implemented on Python.

System Requirements

- Recommended Python version:3.8.x
- or below
- MacOS, Windows (including WSL), and Linux are supported

Installation

This library is available onPyPI

pip install pyband

Basic Usages

Making an oracle request

This section describes the methods used to send a transaction containing an oracle request to BandChain

Step 1: Importpyband and create a parameter:grpc_url with the requiredendpoint which can be foundhere . Then the client instance needs to be initialized in order to allow for the methods in client module to be used.

from pyband . client import Client def

main ( ) :

Step 1

grpc_url

""

ex.laozi-testnet6.bandchain.org(without https://)

c

Client (grpc_url)

if **name** ==

"**main**" : main ( ) Step 2: As the sender's address is required for sending a transaction, we will have to initialize the address first. To do this, start by importingPrivateKey from wallet module. In this example, we will get Mnemonic from environment variables.

from pyband . wallet import PrivateKey

MNEMONIC

os . getenv ("MNEMONIC") private_key = PrivateKey . from_mnemonic (MNEMONIC) public_key = private_key . to_public_key ( ) sender_addr = public_key . to_address ( ) sender = sender_addr . to_acc_bech32 ( ) After that, we will

transform the private key to a public key, the public key to an address, and an address of typeAddress to an address of typestr .

Step 3: Before constructing a transaction, additional information is needed.

As a transaction requires:

- messages
- sequence
- account_num
- chain_id
- fee
- gas
- memo

to be constructed, we will need to get those.

Messages

In this example, we will useMsgRequestData with the following parameters as our message.

- oracle_script_id
-
- : The oracle script ID.
- calldata
-
- : The calldata from a request (e.g., the hex string representing OBI-encoded value of{"symbols": ["ETH"], "multiplier": 100}
-).
- ask_count
-
- : The number of validator required to process this transaction.
- min_count
-
- : The minimum number of validator required to process this transaction.
- client_id
-
- : Name of the client (can be any name or an empty string).
- fee_limit
- <Coin

- : The fee limit.

- prepare_gas
-
- : The amount of gas used in the preparation stage.
- execute_gas
-
- : The amount of gas used in the execution stage.
- sender
-
- : The sender's address.

from pyband . proto . oracle . v1 . tx_pb2 import MsgRequestData

request_msg

MsgRequestData (oracle_script_id = 37 , calldata = bytes . fromhex ("0000000200000003425443000000034554480000000000000064") , ask_count = 4 , min_count = 3 , client_id = "BandProtocol" , fee_limit = [Coin (amount = "100" , denom = "uband")] , prepare_gas = 50000 , execute_gas = 200000 , sender = sender ,) Instead from using bytes for the calldata, oracle binary encoding (obi) can also be used.

from pyband . obi import PyObi

obi

PyObi ("{symbols:[string],multiplier:u64}/{rates:[u64]}") calldata = obi . encode ({ "symbols" :

["ETH"] ,

"multiplier" :

100 }) The message can be any message as listed in Oracle Modules or Cosmos Based Messages . However, please note that our message should be imported from the generated protobuf files .

Sequence and Account Number

Sequence and account number can be retrieved by calling get_account from the client module created instep 1 .

account

c . get_account (sender) account_num = account . account_number sequence = account . sequence

Fee

Fee can be created by using Coin from the generated protobuf file.

from pyband . proto . cosmos . base . v1beta1 . coin_pb2 import Coin

fee

[Coin (amount = "0" , denom = "uband")] Step 4: Now we can construct a Transaction from the transaction module.

from pyband . transaction import Transaction

txn

(Transaction ( ) . with_messages (request_msg) . with_sequence (sequence) . with_account_num (account_num) . with_chain_id (chain_id) . with_gas (2000000) . with_fee (fee) . with_memo ("")) Step 5: Preparing the transaction before sending

Call get_sign_doc to get a signed transaction which we can use to get the signature from.

After that, we can get the raw transaction by calling get_tx_data and putting the signature and public key as the parameters.

sign_doc

txn . get_sign_doc (public_key)

Need to serialize sign_doc of type cosmos_tx_type.SignDoc to string

signature

private_key . sign (sign_doc . SerializeToString ( )) tx_raw_bytes = txn . get_tx_data (signature , public_key) Step 6: After getting the raw transaction, the transaction can now be sent.

While there are 3 modes for sending the transaction, Block mode will be used in this example. We can call send_tx_block_mode with the raw transaction as parameter.

import os

from pyband . client import Client from pyband . transaction import Transaction from pyband . wallet import PrivateKey

from pyband . proto . cosmos . base . v1beta1 . coin_pb2 import Coin from pyband . proto . oracle . v1 . tx_pb2 import MsgRequestData from google . protobuf . json_format import MessageToJson

def

main ( ) :

Step 1

grpc_url

""

ex.laozi-testnet6.bandchain.org(without https://)

c

Client (grpc_url)

Step 2

MNEMONIC

os . getenv ("MNEMONIC") private_key = PrivateKey . from_mnemonic (MNEMONIC) public_key = private_key . to_public_key ( ) sender_addr = public_key . to_address ( ) sender = sender_addr . to_acc_bech32 ( )

Step 3

request_msg

MsgRequestData (oracle_script_id = 37 , calldata = bytes . fromhex ("0000000200000003425443000000034554480000000000000064") , ask_count = 4 , min_count = 3 , client_id = "BandProtocol" , fee_limit = [Coin (amount = "100" , denom = "uband")] , prepare_gas = 50000 , execute_gas = 200000 , sender = sender ,)

account

c . get_account (sender) account_num = account . account_number sequence = account . sequence

fee

[Coin (amount = "0" , denom = "uband")] chain_id = c . get_chain_id ( )

Step 4

txn

(Transaction ( ) . with_messages (request_msg) . with_sequence (sequence) . with_account_num (account_num) . with_chain_id (chain_id) . with_gas (2000000) . with_fee (fee) . with_memo (""))

Step 5

sign_doc

txn . get_sign_doc (public_key) signature = private_key . sign (sign_doc . SerializeToString ( )) tx_raw_bytes = txn . get_tx_data (signature , public_key)

Step 6

tx_block

c . send_tx_block_mode (tx_raw_bytes) print (MessageToJson (tx_block))

if **name** ==

"**main**" : main ( ) And the result should look like this.

{ "height" :

"603247" , "txhash" :

"587FF6D48E5CB8A23715389FE3CAC10262777B395E4D0C554916127461F63446" , "data" :

"0A090A0772657175657374" , "rawLog" :

"[{\"events\":[{\"type\":\"message\",\"attributes\":[{\"key\":\"action\",\"value\":\"request\"}]},{\"type\":\"raw_request\",\"attributes\":
[{\"key\":\"data_source_id\",\"value\":\"61\"},
{\"key\":\"data_source_hash\",\"value\":\"07be7bd61667327aae10b7a13a542c7dfba31b8f4c52b0b60bf9c7b11b1a72ef\"},
{\"key\":\"external_id\",\"value\":\"6\"},{\"key\":\"calldata\",\"value\":\"BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"57\"},
{\"key\":\"data_source_hash\",\"value\":\"61b369daa5c0918020a52165f6c7662d5b9c1eee915025cb3d2b9947a26e48c7\"},
{\"key\":\"external_id\",\"value\":\"0\"},{\"key\":\"calldata\",\"value\":\"BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"62\"},
{\"key\":\"data_source_hash\",\"value\":\"107048da9dbf7960c79fb20e0585e080bb9be07d42a1ce09c5479bbada8d0289\"},
{\"key\":\"external_id\",\"value\":\"3\"},{\"key\":\"calldata\",\"value\":\"BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"60\"},
{\"key\":\"data_source_hash\",\"value\":\"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac\"},
{\"key\":\"external_id\",\"value\":\"5\"},{\"key\":\"calldata\",\"value\":\"huobipro BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"59\"},
{\"key\":\"data_source_hash\",\"value\":\"5c011454981c473af3bf6ef93c76b36bfb6cc0ce5310a70a1ba569de3fc0c15d\"},
{\"key\":\"external_id\",\"value\":\"2\"},{\"key\":\"calldata\",\"value\":\"BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"60\"},
{\"key\":\"data_source_hash\",\"value\":\"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac\"},
{\"key\":\"external_id\",\"value\":\"4\"},{\"key\":\"calldata\",\"value\":\"binance BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"60\"},
{\"key\":\"data_source_hash\",\"value\":\"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac\"},
{\"key\":\"external_id\",\"value\":\"9\"},{\"key\":\"calldata\",\"value\":\"bittrex BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"60\"},
{\"key\":\"data_source_hash\",\"value\":\"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac\"},
{\"key\":\"external_id\",\"value\":\"7\"},{\"key\":\"calldata\",\"value\":\"kraken BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"60\"},
{\"key\":\"data_source_hash\",\"value\":\"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac\"},
{\"key\":\"external_id\",\"value\":\"8\"},{\"key\":\"calldata\",\"value\":\"bitfinex BTC ETH\"},{\"key\":\"fee\"},
{\"key\":\"data_source_id\",\"value\":\"58\"},
{\"key\":\"data_source_hash\",\"value\":\"7e6759fade717a06fb643392bfde837bfc3437da2ded54feed706e6cd35de461\"},
{\"key\":\"external_id\",\"value\":\"1\"},{\"key\":\"calldata\",\"value\":\"BTC ETH\"},{\"key\":\"fee\"}]},
{\"type\":\"request\",\"attributes\":[{\"key\":\"id\",\"value\":\"306633\"},{\"key\":\"client_id\",\"value\":\"BandProtocol\"},
{\"key\":\"oracle_script_id\",\"value\":\"37\"},
{\"key\":\"calldata\",\"value\":\"0000000200000003425443000000034554480000000000000064\"},
{\"key\":\"ask_count\",\"value\":\"4\"},{\"key\":\"min_count\",\"value\":\"3\"},{\"key\":\"gas_used\",\"value\":\"111048\"},
{\"key\":\"total_fees\"},{\"key\":\"validator\",\"value\":\"bandvaloper1zl5925n5u24njn9axpygz8lhjl5a8v4cpkzx5g\"},
{\"key\":\"validator\",\"value\":\"bandvaloper17n5rmujk78nkgss7tjecg4nfzn6geg4cqtyg3u\"},
{\"key\":\"validator\",\"value\":\"bandvaloper1p46uhvdk8vr829v747v85hst3mur2dzlhfemmz\"},
{\"key\":\"validator\",\"value\":\"bandvaloper1ldtwjzsplhxzhrg3k5hhr8v0qterv05vpdxp9f\"}]}]}]" , "logs" :

[{ "events" :

[{ "type" :

"message" , "attributes" :

[{

"key" :

"action" ,

"value" :

"request"

}] } , { "type" :

"raw_request" , "attributes" :

[{

"key" :

"data_source_id" ,

"value" :

"61"

} , { "key" :

"data_source_hash" , "value" :

"07be7bd61667327aae10b7a13a542c7dfba31b8f4c52b0b60bf9c7b11b1a72ef" } , {

"key" :

"external_id" ,

"value" :

"6"

} , {

"key" :

"calldata" ,

"value" :

"BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"57"

} , { "key" :

"data_source_hash" , "value" :

"61b369daa5c0918020a52165f6c7662d5b9c1eee915025cb3d2b9947a26e48c7" } , {

"key" :

"external_id" ,

"value" :

"0"

} , {

"key" :

"calldata" ,

"value" :

"BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"62"

} , { "key" :

"data_source_hash" , "value" :

"107048da9dbf7960c79fb20e0585e080bb9be07d42a1ce09c5479bbada8d0289" } , {

"key" :

"external_id" ,

"value" :

"3"

} , {

"key" :

"calldata" ,

"value" :

"BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"60"

} , { "key" :

"data_source_hash" , "value" :

"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac" } , {

"key" :

"external_id" ,

"value" :

"5"

} , {

"key" :

"calldata" ,

"value" :

"huobipro BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"59"

} , { "key" :

"data_source_hash" , "value" :

"5c011454981c473af3bf6ef93c76b36bfb6cc0ce5310a70a1ba569de3fc0c15d" } , {

"key" :

"external_id" ,

"value" :

"2"

} , {

"key" :

"calldata" ,

"value" :

"BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"60"

} , { "key" :

"data_source_hash" , "value" :

"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac" } , {

"key" :

"external_id" ,

"value" :

"4"

} , {

"key" :

"calldata" ,

"value" :

"binance BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"60"

} , { "key" :

"data_source_hash" , "value" :

"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac" } , {

"key" :

"external_id" ,

"value" :

"9"

} , {

"key" :

"calldata" ,

"value" :

"bittrex BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"60"

} , { "key" :

"data_source_hash" , "value" :

"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac" } , {

"key" :

"external_id" ,

"value" :

"7"

} , {

"key" :

"calldata" ,

"value" :

"kraken BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"60"

} , { "key" :

"data_source_hash" , "value" :

"2e588de76a58338125022bc42b460072300aebbcc4acaf55f91755c1c1799bac" } , {

"key" :

"external_id" ,

"value" :

"8"

} , {

"key" :

"calldata" ,

"value" :

"bitfinex BTC ETH"

} , {

"key" :

"fee"

} , {

"key" :

"data_source_id" ,

"value" :

"58"

} , { "key" :

"data_source_hash" , "value" :

"7e6759fade717a06fb643392bfde837bfc3437da2ded54feed706e6cd35de461" } , {

"key" :

"external_id" ,

"value" :

"1"

} , {

"key" :

"calldata" ,

"value" :

"BTC ETH"

} , {

"key" :

"fee"

}] } , { "type" :

"request" , "attributes" :

[{

"key" :

"id" ,

"value" :

"306633"

} , {

"key" :

"client_id" ,

"value" :

"BandProtocol"

} , {

"key" :

"oracle_script_id" ,

"value" :

"37"

} , { "key" :

"calldata" , "value" :

"0000000200000003425443000000034554480000000000000064" } , {

"key" :

"ask_count" ,

"value" :

"4"

} , {

"key" :

"min_count" ,

"value" :

"3"

} , {

"key" :

"gas_used" ,

"value" :

"111048"

} , {

"key" :

"total_fees"

} , { "key" :

"validator" , "value" :

"bandvaloper1zl5925n5u24njn9axpygz8lhjl5a8v4cpkzx5g" } , { "key" :

"validator" , "value" :

"bandvaloper17n5rmujk78nkgss7tjecg4nfzn6geg4cqtyg3u" } , { "key" :

"validator" , "value" :

"bandvaloper1p46uhvdk8vr829v747v85hst3mur2dzlhfemmz" } , { "key" :

"validator" , "value" :

"bandvaloper1ldtwjzsplhxzhrg3k5hhr8v0qterv05vpdxp9f" }] }] }] , "gasWanted" :

"2000000" , "gasUsed" :

"566496" }

Sending BAND token

The process of sending BAND token is similar tomaking an oracle request , except we will use MsgSend as our message.

TheMsgSend contains the following parameters:

- from_address
-
- : The sender address which as a string.
- to_address
-
- : The receiver address which as a string.
- amount
-
- : The amount of BAND in Coin that you want to send. In this case, we want to send 1 BAND or 1000000 UBAND

from pyband . proto . cosmos . bank . v1beta1 . tx_pb2 import MsgSend

msg

MsgSend (from_address = sender , to_address =

"band1jrhuqrymzt4mnvgw8cvy3s9zhx3jj0dq30qpte" , amount =

[Coin (amount = "100" , denom = "uband")]) The final code should look as shown below.

import os

from pyband . client import Client from pyband . transaction import Transaction from pyband . wallet import PrivateKey

from pyband . proto . cosmos . base . v1beta1 . coin_pb2 import Coin from pyband . proto . cosmos . bank . v1beta1 . tx_pb2 import MsgSend from google . protobuf . json_format import MessageToJson

def

main ( ) :

Step 1

grpc_url

""

ex.laozi-testnet6.bandchain.org(without https://)

c

Client (grpc_url)

Step 2

MNEMONIC

os . getenv ("MNEMONIC") private_key = PrivateKey . from_mnemonic (MNEMONIC) public_key = private_key . to_public_key ( ) sender_addr = public_key . to_address ( ) sender = sender_addr . to_acc_bech32 ( )

Step 3

send_msg

MsgSend (from_address = sender , to_address =

"band1jrhuqrymzt4mnvgw8cvy3s9zhx3jj0dq30qpte" , amount =

[Coin (amount = "1000000" , denom = "uband")])

account

c . get_account (sender) account_num = account . account_number sequence = account . sequence

fee

[Coin (amount = "0" , denom = "uband")] chain_id = c . get_chain_id ( )

Step 4

txn

(Transaction ( ) . with_messages (send_msg) . with_sequence (sequence) . with_account_num (account_num) . with_chain_id (chain_id) . with_gas (2000000) . with_fee (fee) . with_memo (""))

Step 5

sign_doc

txn . get_sign_doc (public_key) signature = private_key . sign (sign_doc . SerializeToString ( )) tx_raw_bytes = txn . get_tx_data (signature , public_key)

Step 6

tx_block

c . send_tx_block_mode (tx_raw_bytes) print (MessageToJson (tx_block))

if **name** ==

"**main**" : main ( ) And the result should look like this.

{ "height" :

"603302" , "txhash" :

"815F488B3F05F2CBDD57C433DBEAF01FBFB06F378716A8ECDF5888095D6F7F7C" , "data" :

"0A060A0473656E64" , "rawLog" :

"[{\"events\":[{\"type\":\"message\",\"attributes\":[{\"key\":\"action\",\"value\":\"send\"},
{\"key\":\"sender\",\"value\":\"band18p27yl962l8283ct7srr5l3g7ydazj07dqrwph\"},{\"key\":\"module\",\"value\":\"bank\"}]},
{\"type\":\"transfer\",\"attributes\":[{\"key\":\"recipient\",\"value\":\"band1jrhuqrymzt4mnvgw8cvy3s9zhx3jj0dq30qpte\"},
{\"key\":\"sender\",\"value\":\"band18p27yl962l8283ct7srr5l3g7ydazj07dqrwph\"},
{\"key\":\"amount\",\"value\":\"1000000uband\"}]}]}]" , "logs" :

[{ "events" :

[{ "type" :

"message" , "attributes" :

[{

"key" :

"action" ,

"value" :

"send"

} , { "key" :

"sender" , "value" :

"band18p27yl962l8283ct7srr5l3g7ydazj07dqrwph" } , {

"key" :

"module" ,

"value" :

"bank"

}] } , { "type" :

"transfer" , "attributes" :

[{ "key" :

"recipient" , "value" :

"band1jrhuqrymzt4mnvgw8cvy3s9zhx3jj0dq30qpte" } , { "key" :

"sender" , "value" :

"band18p27yl962l8283ct7srr5l3g7ydazj07dqrwph" } , {

"key" :

"amount" ,

"value" :

"1000000uband"

}] }] }] , "gasWanted" :

"2000000" , "gasUsed" :

"49029" }

Getting reference data

This section shows an example on how to query data from BandChain. This example queries the standard price reference based on the given symbol pairs, min count, and ask count.

Step 1: Importpyband and create a parameter:grpc_url with the requiredendpoint which can be foundhere . Then the client instance needs to be initialized in order to allow for the methods in client module to be used.

from pyband . client import Client

def

main ( ) :

Step 1

grpc_url

""

ex.laozi-testnet6.bandchain.org(without https://)

if **name** ==

"**main**" : main ( ) Step 2 After importingClient , the functionget_reference_data can now be used to get the latest price.

The function contains the following parameters

- pairs
-
- : list of cryptocurrency pairs
- min_count
-
- : integer of min count
- ask_count
-
- : integer of ask count

from pyband . client import Client

def

main ( ) :

Step 1

grpc_url

""

ex.laozi-testnet6.bandchain.org(without https://)

c

Client (grpc_url)

Step 2

print (c . get_reference_data (["BTC/USD" ,

"ETH/USD"] ,

3 ,

4))

if **name** ==

"**main**" : main ( ) And running the code above should return a result that looks like this.

[ReferencePrice((pair = "BTC/USD") , (rate = 34614.1) , (updated_at = ReferencePriceUpdated((base = 1625655764) , (quote = 1625715134)))) , ReferencePrice((pair = "ETH/USD") , (rate = 2372.53) , (updated_at = ReferencePriceUpdated((base = 1625655764) , (quote = 1625715134))))] Previous Remote Data Source Executor Next Client Module