Common Usage Example

Make an oracle request

This section describes methods to send a transaction of oracle request to BandChain

Step 1: ImportClient from@bandprotocol/bandchain.js and creates a new instance ofgrpcUrl as a parameter and you can get endpoint fromhere . Then initialize the client instance. Every method in client module can now be used.

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const grpcUrl =

"

// ex.https://laozi-testnet6.bandchain.org/grpc-web const client =

new

Client (grpcUrl) Step 2: Specify an account wallet for sending the transaction. This can be done by importPrivateKey fromWallet module and input a 24-words mnemonic string as a seed. For this example, we will use following mnemonic for demonstration.

subject economy equal whisper turn boil guard giraffe stick retreat wealth card only buddy joy leave genuine resemble submit ghost top polar adjust avoid Here is the example on how to get a private key as an account.

import

{

Wallet

}

from

'@bandprotocol/bandchain.js' const

{

PrivateKey

}

=

Wallet

// Step 2.1 const privkey =

PrivateKey . fromMnemonic ('subject economy equal whisper turn boil guard giraffe stick retreat wealth card only buddy joy leave genuine resemble submit ghost top polar adjust avoid') Then, we will use the private key to generate public key and a BAND address, as shown below

const pubkey = privkey . toPubkey ( ) const sender = pubkey . toAddress ( ) . toAccBech32 ( ) Step 3: As we have both an Client instance and an account wallet, we can now construct a transaction by importingTransaction andMsgRequestData .

As the transaction object requires following attributes,

- a list of messages
- account number
- sequence number

- chain ID
- fee

with following optional fields

- memo (default is empty string)

We will firstly construct aMsgRequestData to be included in a list of messages of the transaction. The message requires 9 fields as shown in the example below.

WithinMsgRequestData

import

{

Obi ,

Message ,

Coin

}

from

'@bandprotocol/bandchain.js'

const obi =

new

Obi ('{symbols:[string],multiplier:u64}/{rates:[u64]}') const calldata = obi . encodeInput ({

symbols :

['ETH'] ,

multiplier :

100

})

const oracleScriptId =

37 const askCount =

4 const minCount =

3 const clientId =

'from_bandchain.js'

let feeLimit =

new

Coin ( ) feeLimit . setDenom ('uband') feeLimit . setAmount ('100000')

const prepareGas =

100000 const executeGas =

200000

const requestMessage =

new

Message . MsgRequestData (oracleScriptId , calldata , askCount , minCount , clientId , sender , [feeLimit] , prepareGas , executeGas) After constructedMsgRequestData , we can get other required fields by following methods to constructs a transaction

- Account and Sequence number are automatically gathered from Transaction's withSender
- method.
- Chain ID can be gathered from Client's getChainId
- method.

import

{

Coin ,

Fee ,

Transaction

}

from

'@bandprotocol/bandchain.js'

let feeCoin =

new

Coin ( ) feeCoin . setDenom ('uband') feeCoin . setAmount ('50000')

const fee =

new

Fee ( ) fee . setAmountList ([feeCoin]) fee . setGasLimit (1000000)

const chainId =

await client . getChainId ( ) const txn =

new

Transaction ( ) txn . withMessages (requestMessage) await txn . withSender (client , sender) txn . withChainId (chainId) txn . withFee (fee) txn . withMemo ('') Step 4: Sign and send the transaction

Now, we had an instance of constructed transaction. In order to sign the transaction getSignDoc method in Transaction instance can be used to get serialzed data of the transaction to be used for signing. Then, use PrivateKey 's sign to sign transaction. Finally, use getTxData to include signature and public key to the transaction to get a complete signed transaction.

const signDoc = txn . getSignDoc (pubkey) const signature = privateKey . sign (signDoc) Step 5: Send the signed transaction to Bandchain be using following method of choices

- sendTxBlockMode
- Send the transaction and wait until committed
- sendTxSyncMode
- Send the transaction and wait untilCheckTx
- phase is done
- sendTxAsyncMode
- Send the transaction and immediately returned

For our example, we will use sendTxBlockMode to send the transaction.

The final code should now look like the code below.

import

{

Client ,

Wallet ,

Obi ,

Message ,

Coin ,

Transaction ,

Fee

}

from

'@bandprotocol/bandchain.js'

const grpcUrl =

''

// ex.https://laozi-testnet6.bandchain.org/grpc-web const client =

new

Client (grpcUrl)

async

function

makeRequest ( )

{ // Step 1: Import a private key for signing transaction const

{

PrivateKey

}

=

Wallet const mnemonic =

'test' const privateKey =

PrivateKey . fromMnemonic (mnemonic) const pubkey = privateKey . toPubkey ( ) const sender = pubkey . toAddress ( ) . toAccBech32 ( )

// Step 2.1: Prepare oracle request's properties const obi =

new

Obi ('{symbols:[string],multiplier:u64}/{rates:[u64]}') const calldata = obi . encodeInput ({

symbols :

['ETH'] ,

multiplier :

100

})

const oracleScriptId =

37 const askCount =

4 const minCount =

3 const clientId =

'from_bandchain.js'

let feeLimit =

new

Coin ( ) feeLimit . setDenom ('uband') feeLimit . setAmount ('100000')

const prepareGas =

100000 const executeGas =

200000

// Step 2.2: Create an oracle request message const requestMessage =

new

Message . MsgRequestData (oracleScriptId , calldata , askCount , minCount , clientId , sender , [feeLimit] , prepareGas , executeGas)

let feeCoin =

new

Coin ( ) feeCoin . setDenom ('uband') feeCoin . setAmount ('50000')

// Step 3.1: Construct a transaction const fee =

new

Fee ( ) fee . setAmountList ([feeCoin]) fee . setGasLimit (1000000)

const chainId =

await client . getChainId ( ) const txn =

new

Transaction ( ) txn . withMessages (requestMessage) await txn . withSender (client , sender) txn . withChainId (chainId) txn . withFee (fee) txn . withMemo ('')

// Step 3.2: Sign the transaction using the private key const signDoc = txn . getSignDoc (pubkey) const signature = privateKey . sign (signDoc)

const txRawBytes = txn . getTxData (signature , pubkey)

// Step 4: Broadcast the transaction const sendTx =

await client . sendTxBlockMode (txRawBytes) console . log (sendTx) }

; (async

( )

=>

{ await

makeRequest ( ) }) ( ) After, we run the script above, the result should look like this. The following log contains logs, which have events related to sent request.

{ "height" : 438884 , "txhash" : "313DBAD237E3E672B432D7F9A422EF953EA42E1029F3562C9EE2AEFB70E7D5A3" , "codespace" : "" , "code" : 0 , "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\":\"07be7bd61667327aae10b7a13..." , "logsList" : [{ "msgIndex" : 0 , "log" : "" , "eventsList" :

[{ "type" : "message" , "attributesList" : [{ "key" : "action" , "value" : "request" }] } , { "type" : "raw_request" , "attributesList" : [{ "key" : "data_source_id" , "value" : "61" } , { "key" : "data_source_hash" , "value" : "07be7bd61667327aae10b7a13a542c7dfba31b8f4c52b0b60bf9c7b11b1a72ef" } , { "key" : "external_id" , "value" : "6" } , { "key" : "calldata" , "value" : "BTC ETH" } , { "key" : "fee" , "value" : "" } , { "key" : "data_source_id" , "value" : "57" } , { "key" : "data_source_hash" , "value" : "61b369daa5c0918020a52165f6c7662d5b9c1eee915025cb3d2b9947a26e48c7" } , { "key" : "external_id" , "value" : "0" } , { "key" : "calldata" , "value" : "BTC ETH" } , { "key" : "fee" , "value" : "" } , { "key" : "data_source_id" , "value" : "62" } , { "key" : "data_source_hash" , "value" : "107048da9dbf7960c79fb20e0585e080bb9be07d42a1ce09c5479bbada8d0289" } , { "key" : "external_id" , "value" : "3" } , { "key" : "calldata" , "value" : "BTC ETH" } , { "key" : "fee" , "value" : "" } , { "key" : "data_source_id" , "value" : "60" } , ... , { "key" : "calldata" , "value" : "BTC ETH" } , { "key" : "fee" , "value" : "" }] } , { "type" : "request" , "attributesList" : [{ "key" : "id" , "value" : "74473" } , { "key" : "client_id" , "value" : "from_bandchain.js" } , { "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" , "value" : "" } , { "key" : "validator" , "value" : "bandvaloper1p46uhvdk8vr829v747v85hst3mur2dzlhfemmz" } , { "key" : "validator" , "value" : "bandvaloper1v0u0tsptnkcdrju4qlj0hswqhnqcn47d20prfy" } , { "key" : "validator" , "value" : "bandvaloper17n5rmujk78nkgss7tjecg4nfzn6geg4cqtyg3u" } , { "key" : "validator" , "value" : "bandvaloper19eu9g3gka6rxlevkjlvjq7s6c498tejnwxjwxx" }] }] }] , "info" : "" , "gasWanted" : 1500000 , "gasUsed" : 660549 , "timestamp" : "" }

Send BAND token

Sending BAND token has code pattern similar to the previous section, except that different type of message is used.

The message used for this section isMsgSend which can be used as shown below

const receiver =

'band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f' const sendAmount =

new

Coin ( ) sendAmount . setDenom ('uband') sendAmount . setAmount ('10') const msg =

new

MsgSend (sender , receiver ,

[sendAmount]) Therefore, final result is as shown follow

import

{

Client ,

Wallet ,

Transaction ,

Message ,

Coin ,

Fee

}

from

'@bandprotocol/bandchain.js'

const

{

PrivateKey

}

=

Wallet const client =

new

Client (")

// ex.https://laozi-testnet6.bandchain.org/grpc-web

// Step 2.1 import private key based on given mnemonic string const privkey =

PrivateKey . fromMnemonic ('subject economy equal whisper turn boil guard giraffe stick retreat wealth card only buddy joy leave genuine resemble submit ghost top polar adjust avoid') // Step 2.2 prepare public key and its address const pubkey = privkey . toPubkey ( ) const sender = pubkey . toAddress ( ) . toAccBech32 ( )

```
const
sendCoin
=
async
( )
=>
{ // Step 3.1 constructs MsgSend message const
{
MsgSend
}
=
Message
// Here we use different message type, which is MsgSend const receiver =
'band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f' const sendAmount =
new
Coin ( ) sendAmount . setDenom ( 'uband' ) sendAmount . setAmount ( '10' ) const msg =
new
MsgSend ( sender , receiver ,
[ sendAmount ] ) // Step 3.2 constructs a transaction const account =
await client . getAccount ( sender ) const chainId =
'band-laozi-testnet6'
let feeCoin =
new
Coin ( ) feeCoin . setDenom ( 'uband' ) feeCoin . setAmount ( '1000' )
const fee =
new
Fee ( ) fee . setAmountList ( [ feeCoin ] ) fee . setGasLimit ( 1000000 ) const tx =
new
Transaction ( ) . withMessages ( msg ) . withAccountNum ( account . accountNumber ) . withSequence ( account . sequence
) . withChainId ( chainId ) . withFee ( fee )
// Step 4 sign the transaction const txSignData = tx . getSignDoc ( pubkey ) const signature = privkey . sign ( txSignData )
const signedTx = tx . getTxData ( signature , pubkey )
// Step 5 send the transaction const response =
await client . sendTxBlockMode ( signedTx ) console . log ( response ) }
; ( async
( )
=>
{ await
```

sendCoin ( ) }) ( ) The response should be similar to as shown below

{ "height" :

443301 , "txhash" :

"026760F78665C03DD4A8786304E01848A4747F0B62F7DB4B31F93C792B2D3D52" , "codespace" :

"" , "code" :

0 , "data" :

"0A060A0473656E64" , "rawLog" :

"[{\"events\":[{\"type\":\"message\",\"attributes\":[{\"key\":\"action\",\"value\":\"send\"},
{\"key\":\"sender\",\"value\":\"band168ukdplr7nrljaleef8ehpyvfhe4n78hz0shsy\"},{\"key\":\"module\",\"value\":\"bank\"}]},
{\"type\":\"transfer\",\"attributes\":[{\"key\":\"recipient\",\"value\":\"band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f\"},
{\"key\":\"sender\",\"value\":\"band168ukdplr7nrljaleef8ehpyvfhe4n78hz0shsy\"},{\"key\":\"amount\",\"value\":\"10uband\"}]}]}]"
, "logsList" :

[{ "msgIndex" :

0 , "log" :

"" , "eventsList" :

[{ "type" :

"message" , "attributesList" :

[{

"key" :

"action" ,

"value" :

"send"

} , { "key" :

"sender" , "value" :

"band168ukdplr7nrljaleef8ehpyvfhe4n78hz0shsy" } , {

"key" :

"module" ,

"value" :

"bank"

}] } , { "type" :

"transfer" , "attributesList" :

[{ "key" :

"recipient" , "value" :

"band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f" } , { "key" :

"sender" , "value" :

"band168ukdplr7nrljaleef8ehpyvfhe4n78hz0shsy" } , {

"key" :

"amount" ,

"value" :

"10uband"

}] }] }] , "info" :

"" , "gasWanted" :

1500000 , "gasUsed" :

49013 , "timestamp" :

"" }

Get reference data

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

Step 1: Importbandchain.js and putgrpc_url_web as a parameter and you can getendpoint fromhere . Then initialize the client instance. Every method in client module can now be used.

import

{

Client

}

from

'@bandprotocol/bandchain.js' // Step 1 const grpcUrl =

"

// ex.https://laozi-testnet6.bandchain.org/grpc-web const client =

new

Client (grpcUrl) Step 2: After we import theClient already, then we call theClient 'sgetReferenceData function to get the latest price

There are 3 parameters

- minCount: Integer of min count
- askCount: Integer of ask count
- pairs: The list of cryprocurrency pairs

The final code should look like the code below.

import

{

Client

}

from

'@bandprotocol/bandchain.js' // Step 1 const grpcUrl =

"

// ex.https://laozi-testnet6.bandchain.org/grpc-web const client =

new

Client (grpcUrl)

// Step 2 const minCount =

3 const askCount =

4 const pairs =

['BTC/USD' ,

'ETH/USD']

; (async

( )

=>

{ console . log (JSON . stringify (await client . getReferenceData (pairs , minCount , askCount))) }) ( ) And the result should look like this.

[{ "pair" :

"BTC/USD" , "rate" :

34078.0954 , "updatedAt" :

{ "base" :

1625579610 , "quote" :

1625579627 } , "requestId" :

{ "base" :

79538 , "quote" :

0 } } , { "pair" :

"ETH/BTC" , "rate" :

0.06759872648278929 , "updatedAt" :

{ "base" :

1625579610 , "quote" :

1625579610 } , "requestId" :

{ "base" :

79538 , "quote" :

79538 } }]

Send BAND token via IBC Transfer

With BandChain built based on the Cosmos-SDK, we also allow interaction with our data oracle throughCosmos Inter-Blockchain-Communication protocol,IBC , which connects other compatible blockchains to request data from BandChain.

To send BAND tokens through IBC Protocol, we will use[MsgTransfer]as a method to represents a message to send coins from one account to another between ICS20 enabled chains. See ICS spechere .

Step 1: First, you need to create aMsgTransfer instance from the Message module. The following code shows you how to create the instance.

import

{

Message

}

from

'@bandprotocol/bandchain.js'

const

{

MsgTransfer

}

=

Message Step 2: Now we can construct theMsgTransfer method, this method requires 5 fields to interact with:

Field Type Description sourcePort string The port on which the packet will be sent sourceChannel string The channel by which the packet will be sent sender string The sender address receiver string The recipient address on the destination chain token Coin The tokens to be transferred timeoutTimestamp number Timeout timestamp (in nanoseconds) relative to the current block timestamp. Your code should look like this.

const receiver =

'band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f' const sourcePort =

'transfer' const sourceChannel =

'channel-25' const sendAmount =

new

Coin ( ) sendAmount . setDenom ('uband') sendAmount . setAmount ('10') const timeoutTimestamp =

moment ( ) . unix ( )

+

600

*

1e9

// timeout in 10 mins

const msg =

new

MsgTransfer (sourcePort , sourceChannel , sendAmount , sender , receiver , timeout_timestamp) Your final code should look something like this:

import

{

Client ,

Wallet ,

Transaction ,

Message ,

Coin ,

Fee

}

from

'@bandprotocol/bandchain.js'

const

{

PrivateKey

}

=

Wallet const client =

new

Client (")

// ex.https://laozi-testnet6.bandchain.org/grpc-web const privkey =

PrivateKey . fromMnemonic ('subject economy equal whisper turn boil guard giraffe stick retreat wealth card only buddy joy leave genuine resemble submit ghost top polar adjust avoid') const pubkey = privkey . toPubkey ( ) const sender = pubkey . toAddress ( ) . toAccBech32 ( )

const

sendCoinIbc

=

async

( )

=>

{ // Step 1 constructs MsgTransfer message const

{

MsgTransfer

}

=

Message

const receiver =

'band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f' const sourcePort =

'transfer' const sourceChannel =

'channel-25' const sendAmount =

new

Coin ( ) sendAmount . setDenom ('uband') sendAmount . setAmount ('10') const timeoutTimestamp =

moment ( ) . unix ( )

+

600

*

1e9

// timeout in 10 mins

const msg =

new

MsgTransfer (sourcePort , sourceChannel , sendAmount , sender , receiver , timeout_timestamp)

// Step 2 constructs a transaction const account =

await client . getAccount (sender) const chainId =

'band-laozi-testnet6'

let feeCoin =

new

Coin ( ) feeCoin . setDenom ('uband') feeCoin . setAmount ('1000')

const fee =

new

Fee ( ) fee . setAmountList ([feeCoin]) fee . setGasLimit (1000000) const tx =

new

Transaction ( ) . withMessages (msg) . withAccountNum (account . accountNumber) . withSequence (account . sequence) . withChainId (chainId) . withFee (fee)

// Step 3 sign the transaction const txSignData = tx . getSignDoc (pubkey) const signature = privkey . sign (txSignData) const signedTx = tx . getTxData (signature , pubkey)

// Step 4 send the transaction const response =

await client . sendTxBlockMode (signedTx) console . log (response) }

; (async

( )

=>

{ await

sendCoinIbc ( ) }) ( )

Connect to your app with Ledger

This tutorial guides you through how to set up and use the Bandchain.js library with your Ledger device to access your Ledger Cosmos (BAND) account(s).

Supported Browsers

- Chrome
- Edge (89.0 and later)
- Opera (76.0 and later)

we strongly recommend using Chrome.

To connect your app you will need to install:

1. Ledger Live
2. The Cosmos Nano App
3. At least one account for Band

Accessing your Ledger

import

{

Wallet

}

from

'@bandprotocol/bandchain.js'

const

{

Ledger

}

=

Wallet

const

connectLedger

=

async

( )

=>

{ const ledger =

await

Ledger . connectLedgerWeb ( ) console . log (ledger) }

; (async

( )

=>

{ await

connectLedger ( ) }) ( )

Sending Band Token using Ledger

To send BAND token using Ledger device, the final code should look something like this:

import

{

Wallet ,

Client ,

Transaction ,

Message ,

Coin ,

Fee

}

from

'@bandprotocol/bandchain.js'

const

sendCoinWithLedger

=

async

( )

=>

```
{ const ledger =
await
Ledger . connectLedgerWeb ( )
const
{
MsgSend
}
=
Message const receiver =
'band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f'
const sendAmount =
new
Coin ( ) sendAmount . setDenom ( 'uband' ) sendAmount . setAmount ( '10' )
const msg =
new
MsgSend ( sender , receiver ,
[ sendAmount ] )
const account =
await client . getAccount ( sender ) const chainId =
'band-laozi-testnet6'
let feeCoin =
new
Coin ( ) feeCoin . setDenom ( 'uband' ) feeCoin . setAmount ( '1000' )
const fee =
new
Fee ( ) fee . setAmountList ( [ feeCoin ] ) fee . setGasLimit ( 1000000 )
const tx =
new
Transaction ( ) . withMessages ( msg ) . withChainId ( chainId ) . withFee ( fee ) . withMemo ( '' )
await tx . withSender ( client , bech32_address )
// Sign a message with Ledger device const signature =
await ledger . sign ( tx ) const signedTx = tx . getTxData ( signature , pubKey ,
127 )
// Create a transaction const response =
await client . sendTxBlockMode ( signedTx ) console . log ( response ) }
; ( async
( )
```

=>

{ await

sendCoinWithLedger ( ) }) ( )

Getting Testnet BAND from Faucet

async

function

getFaucet ( )

{ const body =

{ address :

'band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f' , amount :

'10' , }

let options =

{ method :

'POST' , headers :

{ 'Content-Type' :

'application/json;charset=utf-8' , } , body :

JSON . stringify (body) , }

// See https://docs.bandchain.org/develop/api-endpoints#laozi-testnet-5 let response =

await

fetch ({ BAND_FAUCET_ENDPOINT } , options)

console . log (response) }

getFaucet ( ) // {"txHash": "07EA6C439A72DE3A3FEBD6FC952EBEF54B802CC0A9C00C9A1265561AFE9169A7"} And these are examples of Bandchain.js usages, for more information, feel free to dive into specifications in each module.

Previous Installation Next Client Module