Client Module

This module provides functionalities to query data from BandChain and broadcast transactions to BandChain. It uses gRPC-web behind the scene which interact with gRPC-web proxy server.

Note: The can be found inAPI Endpoints

gRPC Errors

When there are gRPC related errors, that is, gRPC status code is notOK , thePromise result will be rejected with anServiceError object with following fields.

Fields Type Description message string error message code number gRPC status code metadata BrowserHeaders gRPC trailer metadata

getChainId

Get BandChain's Chain ID

Return

- Promise
-
    - Chain ID

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

; (async

( )

=>

{ console . log (await client . getChainId ( )) }) ( ) Result

band-laozi-testnet6

getLatestBlock

Get BandChain's latest block detail

Return

- Block
-
    - BandChain's latest block

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

; (async

( )

=>

{ console . log (await client . getLatestBlock ( )) }) ( ) Result

{ "blockId" :

{ "hash" :

"Di1p0sWqEz/l4aVxlJX0fgVrX5eJYAb5t8cmar45fcg=" , "partSetHeader" :

{ "total" :

1 , "hash" :

"bqKyTzMQd9fnDfS9ldBjf+0FOfynv96YqsalsSQ1f0g=" } } , "block" :

{ "header" :

{ "version" :

{ "block" :

11 , "app" :

0 } , "chainId" :

"band-laozi-testnet2" , "height" :

488306 , "time" :

{ "seconds" :

1625718430 , "nanos" :

770011739 } , "lastBlockId" :

{ "hash" :

"s65ZLJIfoZau9ETSMyqYWTjTCsgB8zFgOMuwOUHhKkU=" , "partSetHeader" :

{ "total" :

1 , "hash" :

"4w05KvzYXCVH84P2uZ6jXduvwo+r/Bc+xhh/454T/Gs=" } } , "lastCommitHash" :

"9nqVW5rgPve3VGg9R8s49DjmsK5/xmG0d6gmGhcWxBQ=" , "dataHash" :

"4ViOMq7cJxVBSCgclZfUqg0k0SbvVM6tj75rhNOfO3Q=" , "validatorsHash" :

"rYFx2BfEhW8duRLFgJ4GZqjXKLH/r95+2Wu3Nn+J1zE=" , "nextValidatorsHash" :

"rYFx2BfEhW8duRLFgJ4GZqjXKLH/r95+2Wu3Nn+J1zE=" , "consensusHash" :

"ek5k0qm1ziK3XpVuICUnTcA7aEbM13JRUqa8DQcn4z4=" , "appHash" :

"00/PMV2HIF+lh69p+q5AHwRt6hqlvo5dtm/blBddWP4=" , "lastResultsHash" :

"VMIWdcTC6ZLJ1gW/VGyZ4yv/X2nis75e2PWnT+fmFoo=" , "evidenceHash" :

"47DEQpj8HBSa+/TlmW+5JCeuQeRkm5NMpJWZG3hSuFU=" , "proposerAddress" :

"2MqI5T0aGLK0h3fNWAcQEKLplio=" } , "data" :

{ "txsList" :

[
"CoYECuUDChgvb3JhY2xlLnYxLk1zZ1JlcG9ydERhdGESyAMIheAIEiQIAhABGh5TTFVHX0FORF9TWU1CT0xfTEVOX05PVF9NQVRDSAoSXAgDGlg0OTguNDUzLDEzNC41MjE0LDAuMTE1NDkyNS
,
"Cu8DCs4DChgvb3JhY2xlLnYxLk1zZ1JlcG9ydERhdGESsQMIhuAIEiQIAhABGh5TTFVHX0FORF9TWU1CT0xfTEVOX05PVF9NQVRDSAoSXggDGlozLjE0OTE5MSw2LjgwNzkxNCwwLjA3ODcxMTMs
,
"CpIDCvECChgvb3JhY2xlLnYxLk1zZ1JlcG9ydERhdGES1AIIh+AIEiQIAhABGh5TTFVHX0FORF9TWU1CT0xfTEVOX05PVF9NQVRDSAoSbggDGmowLjc0MTc0NzkxLDEuMTMwMTE4LDAuMzcwNjE5
, ... ,] } , "evidence" :

{ "evidenceList" :

[ ] } , "lastCommit" :

{ "height" :

488305 , "round" :

0 , "blockId" :

{ "hash" :

"s65ZLJIfoZau9ETSMyqYWTjTCsgB8zFgOMuwOUHhKkU=" , "partSetHeader" :

{ "total" :

1 , "hash" :

"4w05KvzYXCVH84P2uZ6jXduvwo+r/Bc+xhh/454T/Gs=" } } , "signaturesList" :

[{ "blockIdFlag" :

2 , "validatorAddress" :

"Zdyy3QL8E8XZYhDrA8fAUD4m2Jc=" , "timestamp" :

{ "seconds" :

1625718430 , "nanos" :

770011739 } , "signature" :

"mOOkCLg3uHBOUauypHAnWnBmoVITXYrPE/i/AsAMOY4ptpHWdwXD4ZtC8XwOZJ5X1zG3yU3usk2gdvwrw2vbFA==" } , { "blockIdFlag" :

2 , "validatorAddress" :

"xLnySmLJL6Qq4ebq+oPMs+KEerU=" , "timestamp" :

{ "seconds" :

1625718430 , "nanos" :

732447931 } , "signature" :

"Atb6fJN5e2gLThE5gPl+9r9wVdmNhYlyTWXYsgwgs8wb4shgRIdGIMNg4hla/0udzStcvOEy7cO4npYUogGruA==" } , { "blockIdFlag" :

2 , "validatorAddress" :

"D3OpjoewGqrIf2g+qADd6sKpM24=" , "timestamp" :

{ "seconds" :

1625718430 , "nanos" :

825672211 } , "signature" :

"vMp3Q8xo9xLEs+PqyyN+t+rgIHzt4jMTQ0jOpFu1ISFbLBy0VDhIgb2QGaiCR1uyFPVgVVWzJ/hW7NqoTJR7Ug==" } , ... ,] } } }

getAccount(address)

Get BandChain's account information

Parameter

- address
- string
-
    - A bech32-encoded account address

Return

- BaseAccount
-

- An object containing account information

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js' const client =

new

Client ('')

; (async

( )

=>

{ console . log (JSON . stringify (await client . getAccount ('band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f'))) }) ( ) Result

{ "address" :

"band1p46uhvdk8vr829v747v85hst3mur2dzlmlac7f" , "accountNumber" :

242 , "sequence" :

0 }

getAllBalances(address)

Returns all the account balances for the given account address.

Parameter

- address
- string
-
  - A bech32-encoded account address

Return

- Coin[]
-
  - A list of Coin that the account have

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js' const client =

new

Client ('')

; (async

( )

=>

{ console . log (JSON . stringify (await client . getAllBalances ('band1mrdmxkhtr3rgfzfgrkxy5pvjtvnm5qq0my5m0x'))) }) ( ) Result

[{ "amount" :

"10000000" , "denom" :

"uband" }]

getDataSource(id)

Get data source metadata by given ID

Parameter

- id
- number
-
  - Data source ID

Return

- DataSource
-
  - An object containing data source metadata

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

const id =

1

; (async

( )

=>

{ console . log (JSON . stringify (await client . getDataSource (id))) }) ( ) Result

{ "owner" :

"band1jfdmjkxs3hvddsf4ef2wmsmte3s5llqhxqgcfe" , "name" :

"DS1" , "description" :

"TBD" , "filename" :

"32ee6262d4a615f2c3ca0589c1c1af79212f24823453cb3f4cfff85b8d338045" , "treasury" :

"band1jfdmjkxs3hvddsf4ef2wmsmte3s5llqhxqgcfe" , "feeList" :

[ ] }

getOracleScript(id)

Get oracle script metadata by given ID

Parameter

- id
- number
-
    - Oracle Script ID

Return

- OracleScript
-
    - Oracle Script metadata

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

const id =

1

; (async

( )

=>

{ console . log (JSON . stringify (await client . getOracleScript (id))) }) ( ) Result

{ "owner" :

"band1jfdmjkxs3hvddsf4ef2wmsmte3s5llqhxqgcfe" , "name" :

"OS1" , "description" :

"TBD" , "filename" :

"f86b37dbe62c3b8c86ae28523bf09e9963a6b2951dd1a5be79f29f66d8236abf" , "schema" :

"{gas_option:string}/{gweix10:u64}" , "sourceCodeUrl" :

"" }

getRequestByID(id)

Get an oracle request by given request ID

Parameter

- id
- number
-
    - Request ID

Return

- Request
-
    - Information of the oracle request

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

const id =

143959

; (async

( )

=>

{ console . log (JSON . stringify (await client . getRequestById (id))) }) ( ) Result

{ "request" :

{ "oracleScriptId" :

37 , "calldata" :

"AAAADwAAAANVTkkAAAAFU1VTSEkAAAAEVVNEQwAAAARVU0RUAAAAA0RBSQAAAANZRkkAAAADU05YAAAABFNVU0QAAAADTUtSAAAAA0NSVgAAAAZSRU5CVEMAAAAEV0JUQwAAAAF
, "requestedValidatorsList" :

["bandvaloper17n5rmujk78nkgss7tjecg4nfzn6geg4cqtyg3u" , "bandvaloper1p46uhvdk8vr829v747v85hst3mur2dzlhfemmz" , "bandvaloper1274qgg28xkz6f3upx05ftr9zepgmtfgts392dy" ,
"bandvaloper1lm2puy995yt8dh53cnazk3ge3m27t7cay4ndaq" , "bandvaloper1v0u0tsptnkcdrju4qlj0hswqhnqcn47d20prfy" , "bandvaloper1a570h9e3rtvfhm030ta5hvel7e7e4lh4pgv8wj"] , "minCount" :

3 , "requestHeight" :

488761 , "requestTime" :

1625719798 , "clientId" :

"alpha" , "rawRequestsList" :

[{ "externalId" :

6 , "dataSourceId" :

61 , "calldata" :

"REFJIExJTksgQ09NUA==" } , { "externalId" :

3 , "dataSourceId" :

62 , "calldata" :

"VU5JIFNVU0hJIFVTREMgVVNEVCBEQUkgWUZJIFNOWCBTVVNEIE1LUiBDUlYgUkVOQlRDIFdCVEMgTElOSyBDT01QIEJBTkQ=" } , { "externalId" :

0 , "dataSourceId" :

57 , "calldata" :

"VVNEVCBCQU5E" } , ... ,] , "executeGas" :

1000000 } , "reportsList" :

[{ "validator" :

"bandvaloper1p46uhvdk8vr829v747v85hst3mur2dzlhfemmz" , "inBeforeResolve" :

true , "rawReportsList" :

[{ "externalId" :

2 , "exitCode" :

1 , "data" :

"NDI5IENsaWVudCBFcnJvcjogVG9vIE1hbnkgUmVxdWVzdHMgZm9yIHVybDogaHR0cHM6Ly9hcGkuY29pbmdlY2tvLmNvbS9hcGkvdjMvY29pbnMvbGlzdAo=" } , { "externalId" :

0 , "exitCode" :

0 , "data" :

"MS4wMDExODUsNi40NjU4MzcK" } , { "externalId" :

1 , "exitCode" :

0 , "data" :

"MjEuMDksOC4zNjksMC45OTk4LDEsMS4wMDEsMzQzMTYuNzMsMTAuMjEsMS4wMDUsMjcxOS41NCwxLjg2OCwzMzIxOS42NSwxOS4wNSw0MjMuOSw2LjQwMgo=" } , ... ,] } , { "validator" :

"bandvaloper1274qgg28xkz6f3upx05ftr9zepgmtfgts392dy" , "inBeforeResolve" :

true , "rawReportsList" :

[{ "externalId" :

0 , "exitCode" :

0 , "data" :

"MS4wMDExODUsNi40NjU4MzcK" } , { "externalId" :

6 , "exitCode" :

0 , "data" :

"MS4wMDA4NTMsMTkuMTY0NjIsNDI2LjYK" } , { "externalId" :

1 , "exitCode" :

0 , "data" :

"MjEuMDksOC4zNjksMC45OTk4LDEsMS4wMDEsMzQzMTYuNzMsMTAuMjEsMS4wMDUsMjcxOS41NCwxLjg2OCwzMzIxOS42NSwxOS4wNSw0MjMuOSw2LjQwMgo=" } , ... ,] } , { "validator" :

"bandvaloper1v0u0tsptnkcdrju4qlj0hswqhnqcn47d20prfy" , "inBeforeResolve" :

true , "rawReportsList" :

[{ "externalId" :

0 , "exitCode" :

0 , "data" :

"MS4wMDExODUsNi40NjU4MzcK" } , { "externalId" :

1 , "exitCode" :

0 , "data" :

"MjEuMDksOC4zNjksMC45OTk4LDEsMS4wMDEsMzQzMTYuNzMsMTAuMjEsMS4wMDUsMjcxOS41NCwxLjg2OCwzMzIxOS42NSwxOS4wNSw0MjMuOSw2LjQwMgo=" } , { "externalId" :

3 , "exitCode" :

0 , "data" :

"MjEuMjUyMSw4LjQ0MjY4OSwxLjAwMDkxMSwxLjAwMTA0MSwxLjAwMDgzLDM0NTQ4LjUxOTEsMTAuMTMwNSwxLjAwNjU2NywyNzMxLjA3TgsMS44ODk5MTMsMzMyMDMuMzkwMSwzMzI3Mi4
} , ... ,] } ,] , "result" :

{ "clientId" :

"alpha" , "oracleScriptId" :

37 , "calldata" :

"AAAADwAAAANVTkkAAAAFU1VTSEkAAAAEVVNEQwAAAARVU0RUAAAAA0RBSQAAAANZRkkAAAADU05YAAAABFNVU0QAAAADTUtSAAAAA0NSVgAAAAZSRU5CVEMAAAAEV0JUQwAAAAF
, "askCount" :

6 , "minCount" :

3 , "requestId" :

143959 , "ansCount" :

6 , "requestTime" :

1625719798 , "resolveTime" :

1625719807 , "resolveStatus" :

1 , "result" :

"AAAADwAAAAT t3DGgAAAAAfZv9mAAAAAAO5oGsAAAAAA7msoAAAAAADumO7AAAB9s/0p6AAAAAAJckF8gAAAAADvnFT8AAAJ8Y5ZhgAAAAABwpcioAAAeNwdVEQ8AAB45MfNWAAAAAAR1h
} }

getReporters(validator)

Get a list of reporter account addresses associated with given validator

Parameter

- validator
- string
-
    - a bech32-encoded validator address

Return

- string[]
-
    - a list of reporter's bech32-encoded account address

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client ('')

; (async

( )

=>

{ console . log (JSON . stringify (await client . getReporters ('bandvaloper17n5rmujk78nkgss7tjecg4nfzn6geg4cqtyg3u'))) }) ( ) Result

["band17n5rmujk78nkgss7tjecg4nfzn6geg4cvaqt5h" , "band1wc6r20m8qg7p3lze55kzen5uwssdvwr7wl5w4q" , "band1wm0lw8wzt094xdyxx4ukx432q9vcwdl9zmwa4x" ,
"band10ptt5622ezszsvrcum07ehng3merea9x5jetv2" , "band10lyra24wxsme03pe47du6xfurtsqzs99mn5r94" , "band1ek7hfydf3xgz3k6nnsy2zrg0xxuzkvhzrykrn5"]

getLatestRequest(oid, calldata, minCount, askCount)

Search for latest request that match given oracle script ID, calldata, min count, and ask count.

Parameter

- oid
- number
-
    - Oracle script ID
- calldata
- string
-
    - OBI-encoded calldata of the oracle request in hex format
- minCount
- number
-

    - The minimum number of validators necessary for the request to proceed to the execution phase
- askCount
- number
    - The number of validators that are requested to respond to this request

Return

- QueryRequestResponse
    - An object containing oracle request information, reports of the request, and final result

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

const oid =

37 const calldata = '0000000600000003425443000000034554480000000034d4952000000003414e4300000004444f4745000000044c554e41000000003b9aca00' const minCount =

3 const askCount =

4

; (async

( )

=>

{ console . log (JSON . stringify (await client . getLatestRequest (oid , calldata , minCount , askCount))) }) ( ) Result

{ "request" :

{ "oracleScriptId" :

37 , "calldata" :

"AAAABgAAAANCVEMAAAADRVRIAAAAA01JUgAAAANBTkMAAAAERE9HRQAAAARMVU5BAAAAADuaygA=" , "requestedValidatorsList" :

["bandvaloper1lm2puy995yt8dh53cnazk3ge3m27t7cay4ndaq" , "bandvaloper17n5rmujk78nkgss7tjecg4nfzn6geg4cqtyg3u" , "bandvaloper1a570h9e3rtvfhm030ta5hvel7e7e4lh4pgv8wj" , ... ,] , "minCount" :

3 , "requestHeight" :

493003 , "requestTime" :

1625732656 , "clientId" :

"mirror-protocol" , "rawRequestsList" :

[{ "externalId" :

6 , "dataSourceId" :

61 , "calldata" :

"QlRDIEVUSA==" } , { "externalId" :

0 , "dataSourceId" :

57 , "calldata" :

"QlRDIEVUSA==" } , { "externalId" :

3 , "dataSourceId" :

62 , "calldata" :

"QlRDIEVUSCBNSVIgQU5DIERPR0UgTFVOQQ==" } , ... ,] , "executeGas" :

1000000 } , "reportsList" :

[{ "validator" :

"bandvaloper1t9vedyzsxewe6lhpf9vm47em2hly23xm6uqtec" , "inBeforeResolve" :

true , "rawReportsList" :

[{ "externalId" :

6 , "exitCode" :

0 , "data" :

"MzI0NDQuMzMsMjE3Mi4yNAo=" } , { "externalId" :

2 , "exitCode" :

0 , "data" :

"MzI1MzQsMjE4MS42MSwzLjc2LDIuMTcsMC4yMTMzMDUsNi41OAo=" } , { "externalId" :

1 , "exitCode" :

0 , "data" :

"MzI0NDAuMDIsMjE3Ni44MywyLjE2MSwwLjIxMjMsNi41NzEK" } , ... ,] } , { "validator" :

"bandvaloper1a570h9e3rtvfhm030ta5hvel7e7e4lh4pgv8wj" , "inBeforeResolve" :

true , "rawReportsList" :

[{ "externalId" :

6 , "exitCode" :

0 , "data" :

"MzI0NDQuMzQsMjE3Mi4yNAo=" } , { "externalId" :

2 , "exitCode" :

0 , "data" :

"MzI1MzIsMjE4MS42MywzLjc2LDIuMTcsMC4yMTMzNjcsNi41OAo=" } , { "externalId" :

3 , "exitCode" :

0 , "data" :

"MzI0NDQuMTA3NywyMTc1Ljk3MTcsMy43NDUxMTUsMi4xNjA5MTQsMC4yMTI1OTk4Miw2LjU5OTAxOAo=" } , ... ,] } , { "validator" :

"bandvaloper1l2hchtyawk9tk43zzjrzr2lcd0zyxngcjdsshe" , "inBeforeResolve" :

true , "rawReportsList" :

[{ "externalId" :

2 , "exitCode" :

0 , "data" :

"MzI1MzIsMjE4MS42MywzLjc2LDIuMTcsMC4yMTMzNjcsNi41OAo=" } , { "externalId" :

0 , "exitCode" :

0 , "data" :

"MzI0NjYuNTg0OCwyMTYzLjA2NjYK" } , { "externalId" :

1 , "exitCode" :

0 , "data" :

"MzI0NDAuMDIsMjE3Ni44MywyLjE2MSwwLjIxMjMsNi41NzEK" } , ... ,] } , ... ,] , "result" :

{ "clientId" :

"mirror-protocol" , "oracleScriptId" :

37 , "calldata" :

"AAAABgAAAANCVEMAAAADRVRIAAAAA01JUgAAAANBTkMAAAAERE9HRQAAAARMVU5BAAAAADuaygA=" , "askCount" :

6 , "minCount" :

3 , "requestId" :

149702 , "ansCount" :

3 , "requestTime" :

1625732656 , "resolveTime" :

1625732662 , "resolveStatus" :

1 , "result" :

"AAAABgAAHYGBsQoQAAAB+mFMa5AAAAAA3znreAAAAACAzj5AAAAAAAyncOAAAAABh6mAwA==" } }

sendTxBlockMode(txBytes)

Send a transaction using block mode, that is, send and wait until the transaction has been committed to a block.

Parameter

- txBytes
- Uint8Array | string
-
    - an byte array of serialized signed transaction

Return

- TxResponse
-
    - An object of transaction response

Example

import

{

Client ,

Wallet ,

Transaction ,

Message ,

Coin ,

Fee

}

from

'@bandprotocol/bandchain.js'

const

{

PrivateKey

}

=

Wallet const client =

new

Client ('')

// 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 . toAny ( )) . 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 (JSON . stringify (response)) }

; (async

( )

=>

{ await

sendCoin ( ) }) ( ) Result

{ "height" :

493527 , "txhash" :

"F76593C2165A42E39464FEAD998AE80970655D82B18085FD65917ACC0979279D" , "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" :

"" }

sendTxSyncMode(txBytes)

Send a transaction in sync mode, that is, send and wait until transaction has passed CheckTx phase.

Parameter

- txBytes
- Uint8Array
-
    - a byte array of serialized signed transaction

Return

- TxResponse
-
    - An object of transaction response

Example

import

{

Client ,

Wallet ,

Transaction ,

Message ,

Coin ,

Fee

}

from

'@bandprotocol/bandchain.js'

const

{

PrivateKey

}

=

Wallet const client =

new

Client (")

// 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 . toAny ( )) . 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 . sendTxSyncMode (signedTx) console . log (JSON . stringify (response)) }

; (async

( )

=>

{ await

sendCoin ( ) }) ( ) Result

{ "height" :

0 , "txhash" :

"48620C4242AFB1F18F0FA1C72ADE42C26FDCC804CB20E2BDBAE8B0097C5900B6" , "codespace" :

"" , "code" :

0 , "data" :

"" , "rawLog" :

"[]" , "logsList" :

[ ] , "info" :

"" , "gasWanted" :

0 , "gasUsed" :

0 , "timestamp" :

"" }

sendTxAsyncMode(data)

Send a transaction in async mode, that is, send and returned immediantly without waiting for the transaction processes.

Parameter

- txBytes
- Uint8Array
-
    - a byte array of serialized signed transaction

Return

- TxResponse
-
    - An object of transaction response

Example

import

{

Client ,

Wallet ,

Transaction ,

Message ,

Coin ,

Fee

}

from

'@bandprotocol/bandchain.js'

const

{

PrivateKey

}

=

Wallet const client =

new

Client ('')

// 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 . toAny ( )) . 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 . sendTxAsyncMode (signedTx) console . log (JSON . stringify (response)) }

; (async

( )

=>

{ await

sendCoin ( ) }) ( ) Result

{ "height" :

0 , "txhash" :

"8A3573AC59BC6CC1A7ECF18A2E1FC50E8AE73E69A68351496872F08186D6158F" , "codespace" :

"" , "code" :

0 , "data" :

"" , "rawLog" :

"" , "logsList" :

[ ] , "info" :

"" , "gasWanted" :

0 , "gasUsed" :

0 , "timestamp" :

"" }

getReferenceData(pairs, minCount, askCount)

Get current prices from standard price references oracle script based on given symbol pairs, min count, and ask count.

Parameter

- pairs
- string[]
-
    - a list of symbol pairs e.g. BTC/USD, ETH/BTC, etc.
- minCount
- number
-
    - The minimum number of validators necessary for the request to proceed to the execution phase
- askCount
- number
-
    - The number of validators that are requested to respond to this request

Return

- [ReferenceData[]
-] - A list of prices for given pairs

Example

import

{

Client

}

from

'@bandprotocol/bandchain.js'

const client =

new

Client (")

; (async

( )

=>

{ console . log (JSON . stringify (await client . getReferenceData (['BTC/USD' ,

'ETH/BTC'] ,

3 ,

4))) }) ( ) Result

[{ "pair" :

"BTC/USD" , "rate" :

32557.06795 , "updatedAt" :

{ "base" :

1625736254 , "quote" :

1625736266 } , "requestId" :

{ "base" :

151316 , "quote" :

0 } } , { "pair" :

"ETH/BTC" , "rate" :

0.06693865225661391 , "updatedAt" :

{ "base" :

1625736254 , "quote" :

1625736254 } , "requestId" :

{ "base" :

151316 , "quote" :

151316 } }]Previous Common Usage Example Next Data Module