Frequently Asked Questions

A list of frequently asked questions can be found below. If you still can't find the answer to your question below, please follow up onDiscord(opens in a new tab)

Is it possible to create a multisig account?

Yes, multisig accounts are natively supported. Here is a simple example of how you can set up a multisig account:

1. Add two accounts to your local keychain

seid

keys

add

ms1 seid

keys

add

ms2 1. Create a multisig account requiring the above 2 accounts

seid

keys

add

ms1ms2

--multisig-threshold=2

--multisig=ms1,ms2

Signing and broadcasting a TX

1. Define an unsigned-tx.json file with your unsigned transaction

{ "body" : { "messages" : [{ "@type" :

"/cosmos.bank.v1beta1.MsgSend" , "from_address" :

"MULTI_SIG_ACCOUNT" , "to_address" :

"DESIRED_DESTINATION_ADDRESS" , "amount" : [{ "denom" :

"usei" , "amount" :

"10" }] }] , "memo" :

"" , "timeout_height" :

"0" , "extension_options" : [] , "non_critical_extension_options" : [] } , "auth_info" : { "signer_infos" : [] , "fee" : { "amount" : [{ "denom" :

"usei" , "amount" :

"100000" }] , "gas_limit" :

"200000" , "payer" :

"" , "granter" :

"" } } , "signatures" : [] } 1. Sign unsigned tx from both signers (see unsigned-tx.json) and output json files of signed tx's

seid

tx

sign

unsigned-tx.json

--multisig=multisigAccountName

--from=ms1

--output-document=signer1_signedTx.json

--node

YOUR_RPC_URL seid

tx

sign

unsigned-tx.json

--multisig=multisigAccountName

--from=ms2

--output-document=signer2_signedTx.json

--node

YOUR_RPC_URL 1. Sign tx from multisig account and output to json file of signed tx

seid

tx

multisign

unsigned-tx.json

ms1ms2

signer1_signedTx.json

signer2_signedTx.json

signedTx.json 1. Broadcast multisig tx

seid

tx

broadcast

signedTx.json

Where can I find examples?

We have some Typescript examples in the documentation for our JS library, SeiJS:

- SeiJS TypeDocs Examples(opens in a new tab)

To see other examples of how to interact with Sei, you can check out our js-examples repo.

- @sei-js/examples(opens in a new tab)

Where can I find chain and asset information

- Sei chain registry(opens in a new tab)

Where can I find current rest and rpc urls?

Public endpoints for all sei chains can be found in theTools and Resources section

Is there a Sei faucet for getting testnet funds?

Yes, you can head to Sei Faucet to connect your wallet and request tokens from the faucet. Instructions are here

Why did smart contract ran out of gas at 300,000?

For all the Sei provided public nodes, we have set a maximum gas of 300,000. If your contract requires more gas, you will need to set up your own node and rpc url.

Additionally it is important that you run all your smart contracts through rust-optimizer so that your gas is low and your contract is optimized.

How can I create a wallet given a mnemonic seed phrase?

If you are using the seid CLI you can run:

seid

keys

add

NAME

--recover

SEED_PHASE to recover the account as a local account. (Replace NAME with anything you want to name this local account and SEED_PHASE with the seed phase.)

In the context of a Javascript application:

import { restoreWallet } from

'@sei-js/core' ;

//Takes an optional account index const

wallet

=

await

restoreWallet (YOUR_SEED_PHRASE ,

2);

// Now your wallet can be used with other @sei-js helper functions const

client

=

await

getSigningCosmWasmClient (RPC_URL , wallet);

Resolving "ReferenceError: process is not defined"

The "ReferenceError: process is not defined" error often occurs when developing web3 applications that rely on Node.js-specific libraries, such as process or Buffer. This error shows up because web browsers do not natively support these Node.js modules. While the @sei-js package internally handles these discrepancies by including the necessary polyfills, you might encounter this or similar errors due to other third-party libraries not bundling those dependencies. When this happens, you'll have to manually include the required polyfills in your project.

If you are using Vite or rollup you can add the following to the entry point of your app.

import

*

as process from

'process' ; import { Buffer } from

'buffer' ;

// Polyfill self for browser and global for Node.js const

globalObject

=

typeof self !==

'undefined'

? self : global;

Object .assign (globalObject , { process : process , Buffer : Buffer }); If you are using Webpack you can use the following plugin in you Webpack config.

yarn

add

-D

node-polyfill-webpack-plugin import NodePolyfillPlugin from

'node-polyfill-webpack-plugin' ;

...

// the rest of your webpack config plugins : [... new

NodePolyfillPlugin () , ...] , ... Last updated onMarch 12, 2024Media Package