Example Usage

The Sei precompiles can be used like any standard smart contract on the EVM. For example, usingethers.js(opens in a new tab) , you can query and execute actions on a CosmWasm contract as follows:

Setup

To installethers , run the following command in your project directory terminal:

npm

install

ethers npm

install

@sei-js/evm Next, you'll need to use one of the precompiles inEVM Precompiles section. In this example, we're going to be using theCosmWasm precompile :

// Import Wasm precompile address and ABI // View the entire ABI here: https://github.com/sei-protocol/sei-chain/tree/evm/precompiles/wasmd import { WASM_PRECOMPILE_ABI , WASM_PRECOMPILE_ADDRESS } from

"@sei-js/evm" ;

Using the contract

Next, we'll set up a provider and contract to interact with the blockchain:

import { ethers , toUtf8Bytes , toUtf8String } from

"ethers" ;

// Using MetaMask as the signer and provider const

provider

=

new

ethers .BrowserProvider (window .ethereum); const

signer

=

await

provider .getSigner ();

// Create a contract with the signer const

contract

=

new

ethers .Contract (WASM_PRECOMPILE_ADDRESS , WASM_PRECOMPILE_ABI , signer); If using MetaMask, the wallet must be switched to the Sei EVM Devnet chain. Learn how to import the Sei EVM Devnet chainhere .

Querying & Executing a CosmWasm Contract

Once you have the contract, you can query and execute messages to any CosmWasm smart contract.

// Counter CosmWasm contract (used for testing on arctic-1) // Replace with your contract as needed const

COUNTER_CONTRACT_ADDRESS

= "sei1eyfccmjm6732k7wp4p6gdjwhxjwsvje44j0hfx8nkgrm8fs7vqfsy2jxff" ;

// Query to get the count on the counter contract const

queryMsg

= { get_count : {} }; const

queryResponse

=

await

contract .query (COUNTER_CONTRACT_ADDRESS , toUtf8Bytes (JSON .stringify (queryMsg))); console .log (toUtf8String (queryResponse));

// Execute message to increment the count on the contract. // No funds are attached since the increment method does not require sei. const

executeMsg

= { increment : {} }; const

executeResponse

=

await

contract .execute (COUNTER_CONTRACT_ADDRESS , toUtf8Bytes (JSON .stringify (executeMsg)) , toUtf8Bytes (JSON .stringify ([])) // Used for sending funds if needed);

// Wait for the transaction to be confirmed await

executeResponse .wait (); console .log (executeResponse);

Executing a payable function

In this example, we execute the 'donate' method on our contract. This is similar to theincrement method, but also receives funds from the user and stores it in the contract.

// Execute a message to donate to the contract. const

executeMsg

= { donate : {} }

// Funds are attached via overrides. This example is specific to ethers.js const

overrides

= { value :

ethers .parseEther ("3.2") ,

// Sending 3.2 sei } const

executeResponse

=

await

contract .execute (COUNTER_CONTRACT_ADDRESS , toUtf8Bytes (JSON .stringify (executeJSON)) , toUtf8Bytes (JSON .stringify ({denom :

"usdc" , amount :

"100" })) ,

// Also send 100 usdc overrides);

await

executeResponse .wait (); const

receipt

=

await

provider .getTransactionReceipt (executionResponse .hash); For payable contracts, Sei amounts have to be sent directly to the contract while other denoms should use thecoins field.

Last updated onJune 3, 2024 Distribution Governance