[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 0 - Getting Started](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Getting Started](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Blockchain 101](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Blockchain History](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Public and Managed Blockchains](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Consensus in Distributed Networks](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Cryptography](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Self-Assessment Quiz](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Go Introduction - First Steps](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Go Basics](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Go Interfaces](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Control Structures in Go](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Arrays and Slices in Go](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Standard Packages in Go](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Concurrency in Go](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Docker Introduction](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Introduction to the Interchain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[The Interchain Ecosystem](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Getting ATOM and Staking It](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[A Blockchain App Architecture](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Accounts](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Transactions](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Messages](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Modules](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Protobuf](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Multistore and Keepers](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[BaseApp](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Queries](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Events](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Context](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Testing](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Relaying with IBC](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Interchain Security](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Bridges](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Migrations](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 1 Quiz](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 2 - First Steps](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[First Steps](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Setup Your Work Environment](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Run a Node, API, and CLI](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Ignite CLI](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Store Object](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create Custom Messages](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create and Save a Game Properly](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Add a Way to Make a Move](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Emit Game Information](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Record the Game Winner](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 2 Exercise](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[What is IBC?](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC Token Transfer](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC Tooling](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[What is CosmJS?](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Your First CosmJS Actions](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Compose Complex Transactions](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Learn to Integrate Keplr](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Put Your Games in Order](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Auto-Expiring Games](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Let Players Set a Wager](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Handle wager payments](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Integration tests](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Incentivize Players](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Help Find a Correct Move](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Understand IBC Denoms](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Go Relayer](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Hermes Relayer](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[CosmJS Advanced](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create Custom Objects](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create Custom Messages](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Get an External GUI](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Backend Script for Game Indexing](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC Deep Dive](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[IBC Application Developer Introduction](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Create a Leaderboard Chain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Run in Production](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Prepare the Software to Run](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Prepare a Validator and Keys](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Prepare Where the Node Starts](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Prepare and Do Migrations](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Simulate Production in Docker](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Tally Player Info After Production](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Simulate a Migration in Docker](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Final Exam](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[What's Next?](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

[Continue Your Interchain Journey](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html)

Docs Version Switcher

On this page

[Script preparation](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#script-preparation)

[Testnet preparation](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#testnet-preparation)

[Add your imports](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#add-your-imports)

[Define your connection](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#define-your-connection)

[Get a balance](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#get-a-balance)

[Get the faucet address](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#get-the-faucet-address)

[Deserialize the transaction](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#deserialize-the-transaction)

[What is this long string?](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#what-is-this-long-string)

[Deserialize the message](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#deserialize-the-message)

[Prepare a signing client](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#prepare-a-signing-client)

[Send tokens](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#send-tokens)

[With a locally started chain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#with-a-locally-started-chain)

[Preparing your keys](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#preparing-your-keys)

[Update your script](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#update-your-script)

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#your-first-cosmjs-actions) **Your First CosmJS Actions**



Take your first steps with CosmJS. Use it to send some simple transactions.   
  
In this section, you will:

* Download and install CosmJS.
* Create a small experiment.
* Prepare a simple testnet.
* Establish your connection.
* Inspect a balance.
* Send transactions.

Now that you know what CosmJS is, you should take your first steps in using it. A basic feature of a Cosmos chain is the ability to send tokens via the bank module. CosmJS naturally offers functions to cover this facility. You are going to:

1. Use an existing test network (testnet) with a key of your own.
2. Run basic CosmJS commands in a script that you run using the CLI.

Additionally, you can choose to:

1. Start a local chain that exposes RPCs instead of using a testnet.
2. Run the same basic CosmJS commands, but for this local chain.

Along the way, you learn the basic CosmJS concepts needed to start interacting with the Interchain Ecosystem.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#script-preparation) Script preparation

A small, ready-made repository exists so you can experiment with CosmJS. Clone it from [here (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox). You need [NodeJs (opens new window)↗](https://nodejs.org/en/download/), or [Docker](https://ida.interchain.io/tutorials/5-docker-intro/), in which to run NodeJs. If you open the folder in [Visual Studio Code (opens new window)↗](https://code.visualstudio.com/Download), the IDE should give you all the coding help you require. In the cloned folder you need to install the required modules:

**Local**

**Docker**



Copy

$ npm install

Copy

$ docker run --rm -it \

-v $(pwd):/sandbox -w /sandbox \

node:lts-slim \

npm install

Create a new file named experiment.ts. In it, put these lines to confirm it works:



Copy

const runAll = async(): Promise<void> => {

console.log("TODO")

}

runAll()

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/file-preparation/experiment.ts" \l "L1-L5" \t "_blank)

To execute, this TypeScript file needs to be compiled into JavaScript before being interpreted by NodeJs. Add this as a run target in package.json:



Copy

...

"scripts": {

+ "experiment": "ts-node experiment.ts"

...

}

...

package.json

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/file-preparation/package.json" \l "L7" \t "_blank)

Confirm that it does what you want:

**Local**

**Docker**



Copy

$ npm run experiment

Copy

$ docker run --rm -it \

-v $(pwd):/sandbox -w /sandbox \

node:lts-slim \

npm run experiment

This returns:



Copy

> cosmjs-sandbox@1.0.0 experiment

> ts-node experiment.ts

TODO

You will soon make this script more meaningful. With the basic script ready, you need to prepare some elements.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#testnet-preparation) Testnet preparation

The Interchain Ecosystem has a number of testnets running. The Cosmos Hub is currently running a [public testnet (opens new window)↗](https://github.com/cosmos/testnets/tree/master/public) for the Theta upgrade that you are connecting to and running your script on. You need to connect to a public node so that you can query information and broadcast transactions. One of the available nodes is:



Copy

RPC: https://rpc.sentry-01.theta-testnet.polypore.xyz

public

[View source→](https://github.com/cosmos/testnets/tree/master/public" \l "endpoints" \t "_blank)

You need a wallet address on the testnet and you must create a 24-word mnemonic in order to do so. CosmJS can generate one for you. Create a new file generate\_mnemonic.ts with the following script:



Copy

import { DirectSecp256k1HdWallet } from "@cosmjs/proto-signing"

const generateKey = async (): Promise<void> => {

const wallet: DirectSecp256k1HdWallet = await DirectSecp256k1HdWallet.generate(24)

process.stdout.write(wallet.mnemonic)

const accounts = await wallet.getAccounts()

console.error("Mnemonic with 1st account:", accounts[0].address)

}

generateKey()

generate\_mnemonic.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/file-preparation/generate_mnemonic.ts" \t "_blank)

Now create a **key** for our imaginary user **Alice**:

**Local**

**Docker**



Copy

$ npx ts-node generate\_mnemonic.ts > testnet.alice.mnemonic.key

Copy

$ docker run --rm -i \

-v $(pwd):/sandbox -w /sandbox \

node:lts-slim \

npx ts-node generate\_mnemonic.ts \

> testnet.alice.mnemonic.key



You likely need to update Node.js to a later version if this fails. Find a guide [here (opens new window)↗](https://phoenixnap.com/kb/update-node-js-version).

When done, it should also tell you the address of the first account:



Copy

Mnemonic with 1st account: cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf

Temporarily keep this address for convenience, although CosmJS can always recalculate it from the mnemonic. Privately examine the file to confirm it contains your 24 words.



Important considerations:

1. process.stdout.write was used to avoid any line return. Be careful not to add any empty lines or any other character in your .key file (this occurs with VSCode under certain conditions). If you add any characters, CosmJS may not be able to parse it.
2. Adjust the .gitignore file to not commit your .key file by mistake:



Copy

node\_modules

+ \*.key

.gitignore

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/file-preparation/.gitignore" \l "L2" \t "_blank)



For your convenience, all the code and files you've added so far are available [here (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/tree/file-preparation) at the file-preparation branch.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#add-your-imports) Add your imports

You need a small, simple interface to a blockchain, one which could eventually have users. Good practice is to refrain from requesting a user address until necessary (e.g. when a user clicks a relevant button). Therefore, in experiment.ts you first use the read-only client. Import it at the top of the file:



Copy

import { StargateClient } from "@cosmjs/stargate"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L1" \t "_blank)



Note that VSCode assists you to auto-complete [StargateClient (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/stargateclient.ts#L139) if you type CTRL-Space inside the {} of the import line.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#define-your-connection) Define your connection

Next, you need to tell the client how to connect to the RPC port of your blockchain:



Copy

+ const rpc = "rpc.sentry-01.theta-testnet.polypore.xyz:26657"

const runAll = async () ...

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L5" \t "_blank)

Inside the runAll function you [initialize the connection (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/stargateclient.ts#L146) and immediately [check (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/stargateclient.ts#L194) you connected to the right place:



Copy

const runAll = async(): Promise<void> => {

- console.log(TODO)

+ const client = await StargateClient.connect(rpc)

+ console.log("With client, chain id:", await client.getChainId(), ", height:", await client.getHeight())

}

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L8-L9" \t "_blank)

Run again to check with npm run experiment, and you get:



Copy

With client, chain id: theta-testnet-001 , height: 9507032

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#get-a-balance) Get a balance

Normally you would not yet have access to your user's address. However, for this exercise you need to know how many tokens Alice has, so add a temporary new command inside runAll:



Copy

console.log("With client, chain id:"...)

+ console.log(

+ "Alice balances:",

+ await client.getAllBalances("cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf"), // <-- replace with your generated address

+ )

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L10-L13" \t "_blank)

getAllBalances is used because the default token name is not yet known. When you run it again, you get:



Copy

Alice balances: []

If you just created this account, Alice's balance is zero. Alice needs tokens to be able to send transactions and participate in the network. A common practice with testnets is to expose **faucets** (services that send you test tokens for free, within limits).

The Cosmos Hub Testnet faucet has a dedicated [Discord channel (opens new window)↗](https://discord.com/channels/669268347736686612/953697793476821092/958291295741313024) where you can ask for tokens once per day *per Discord user*.

Go to the faucet channel and request tokens for Alice by entering this command in the channel:



Copy

$request [Alice's address] theta

// For example:

$request cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf theta

The faucet bot replies with a link to the transaction from the block explorer:



Copy

✅ https://explorer.theta-testnet.polypore.xyz/transactions/540484BDD342702F196F84C2FD42D63FA77F74B26A8D7383FAA5AB46E4114A9B

Check that Alice received the tokens with npm run experiment, which should return:



Copy

Alice balances: [ { denom: 'uatom', amount: '10000000' } ]

uatom is the indivisible token unit on the Testnet. It is short for micro-ATOM, or µ-ATOM. So 10 million uatom equal 10 ATOM. After this confirmation you can comment out the balance query.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#get-the-faucet-address) Get the faucet address

As an exercise you want Alice to send some tokens back to the faucet, so you need its address. You can request this from the faucet bot, but it is also possible to get it using the transaction hash in experiment.ts.

First you need to get the transaction.

Add the necessary import at the top:



Copy

- import { StargateClient } from "@cosmjs/stargate"

+ import { IndexedTx, StargateClient } from "@cosmjs/stargate"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L1" \t "_blank)

Then, make sure you replace the hash with the one you received from the faucet bot.



Copy

console.log("Alice balances:", ...)

+ const faucetTx: IndexedTx = (await client.getTx(

+ "540484BDD342702F196F84C2FD42D63FA77F74B26A8D7383FAA5AB46E4114A9B",

+ ))!

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L14-L16" \t "_blank)

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#deserialize-the-transaction) Deserialize the transaction

What does faucetTx contain? Add the following line to find out:



Copy

const faucetTx: IndexedTx = ...

+ console.log("Faucet Tx:", faucetTx)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L17" \t "_blank)

Then run the script again using npm run experiment. You find the following output, which in your case contains different values:



Copy

Faucet Tx: {

height: 9487785,

hash: '540484BDD342702F196F84C2FD42D63FA77F74B26A8D7383FAA5AB46E4114A9B',

code: 0,

rawLog: '[{"events":[{"type":"coin\_received","attributes":[{"key":"receiver","value":"cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf"},{"key":"amount","value":"10000000uatom"}]},{"type":"coin\_spent","attributes":[{"key":"spender","value":"cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"},{"key":"amount","value":"10000000uatom"}]},{"type":"message","attributes":[{"key":"action","value":"/cosmos.bank.v1beta1.MsgSend"},{"key":"sender","value":"cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"},{"key":"module","value":"bank"}]},{"type":"transfer","attributes":[{"key":"recipient","value":"cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf"},{"key":"sender","value":"cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"},{"key":"amount","value":"10000000uatom"}]}]}]',

tx: Uint8Array(321) [

10, 148, 1, 10, 145, 1, 10, 28, 47, 99, 111, 115,

109, 111, 115, 46, 98, 97, 110, 107, 46, 118, 49, 98,

101, 116, 97, 49, 46, 77, 115, 103, 83, 101, 110, 100,

18, 113, 10, 45, 99, 111, 115, 109, 111, 115, 49, 53,

97, 112, 116, 100, 113, 109, 109, 55, 100, 100, 103, 116,

99, 114, 106, 118, 99, 53, 104, 115, 57, 56, 56, 114,

108, 114, 107, 122, 101, 52, 48, 108, 52, 113, 48, 104,

101, 18, 45, 99, 111, 115, 109, 111, 115, 49, 55, 116,

118, 100, 52, 104,

... 221 more items

],

gasUsed: 76657,

gasWanted: 200000

}

The structure of this output is JSON. There is a serialized faucetTx.tx of the type Uint8Array. The serialized transaction are the bytes (i.e. Uint8) of the actual transaction that was sent over the testnet by the faucet. It is unintelligible to humans until you deserialize it properly. Use the methods offered by cosmjs-types [Tx (opens new window)↗](https://github.com/confio/cosmjs-types/blob/v0.4.1/src/cosmos/tx/v1beta1/tx.ts#L230) to deserialize it.

Add the necessary import at the top:



Copy

import { IndexedTx, StargateClient } from "@cosmjs/stargate"

+ import { Tx } from "cosmjs-types/cosmos/tx/v1beta1/tx"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L3" \t "_blank)

Then deserialize the transaction:



Copy

console.log("Faucet Tx:", faucetTx)

+ const decodedTx: Tx = Tx.decode(faucetTx.tx)

+ console.log("DecodedTx:", decodedTx)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L18-L19" \t "_blank)

Which, on your next npm run experiment, prints:



Copy

DecodedTx: {

signatures: [

Uint8Array(64) [

106, 244, 26, 232, 175, 96, 235, 168, 96, 55, 157,

222, 49, 142, 64, 207, 67, 109, 40, 45, 153, 232,

112, 134, 251, 97, 72, 162, 169, 62, 245, 134, 59,

241, 75, 31, 146, 11, 176, 159, 185, 41, 100, 171,

175, 78, 120, 186, 24, 136, 103, 160, 205, 64, 180,

131, 9, 137, 178, 221, 68, 28, 122, 169

]

],

body: {

memo: '',

timeoutHeight: Long { low: 0, high: 0, unsigned: true },

messages: [ [Object] ],

extensionOptions: [],

nonCriticalExtensionOptions: []

},

authInfo: {

signerInfos: [ [Object] ],

fee: { gasLimit: [Long], payer: '', granter: '', amount: [Array] }

}

}

The faucet address information you are looking for is inside the body.messages, and must be printed. Add:



Copy

console.log("DecodedTx:", decodedTx)

+ console.log("Decoded messages:", decodedTx.body!.messages)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L20" \t "_blank)

Which, on your next npm run experiment, prints:



Copy

Decoded messages: [

{

typeUrl: '/cosmos.bank.v1beta1.MsgSend',

value: Uint8Array(113) [

10, 45, 99, 111, 115, 109, 111, 115, 49, 53, 97, 112,

116, 100, 113, 109, 109, 55, 100, 100, 103, 116, 99, 114,

106, 118, 99, 53, 104, 115, 57, 56, 56, 114, 108, 114,

107, 122, 101, 52, 48, 108, 52, 113, 48, 104, 101, 18,

45, 99, 111, 115, 109, 111, 115, 49, 55, 116, 118, 100,

52, 104, 99, 115, 122, 113, 55, 108, 99, 120, 117, 119,

122, 114, 113, 107, 101, 112, 117, 97, 117, 57, 102, 121,

101, 51, 100, 97, 108, 54, 48, 54, 122, 102, 26, 17,

10, 5, 117, 97,

... 13 more items

]

}

]

Deserializing the transaction has not fully deserialized any messages that it contains, nor their value, which is again a Uint8Array. The transaction deserializer knows how to properly decode any transaction, but it does not know how to do the same for messages. Messages can in fact be of any type, and each type has its own deserializer. This is not something that the Tx.decode transaction deserializer function knows.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#what-is-this-long-string) What is this long string?

Note the typeUrl: "/cosmos.bank.v1beta1.MsgSend" string. This comes from the [Protobuf](https://ida.interchain.io/academy/2-cosmos-concepts/6-protobuf.html) definitions and is a mixture of:

1. The package where MsgSend is initially declared:



Copy

package cosmos.bank.v1beta1;

proto /

cosmos /

... /

v1beta1 /

tx.proto

[View source→](https://github.com/cosmos/cosmos-sdk/blob/3a1027c/proto/cosmos/bank/v1beta1/tx.proto" \l "L2" \t "_blank)

1. And the name of the message itself, MsgSend:



Copy

message MsgSend {

...

}

proto /

cosmos /

... /

v1beta1 /

tx.proto

[View source→](https://github.com/cosmos/cosmos-sdk/blob/3a1027c/proto/cosmos/bank/v1beta1/tx.proto" \l "L22" \t "_blank)

This typeUrl string you see in the decoded message is the canonical identifier of the type of message that's serialized under the value array. There are many different message types, each coming from different modules or base layers from the Cosmos SDK, and you can find an overview of them [here (opens new window)↗](https://buf.build/cosmos/cosmos-sdk).

The blockchain client itself knows how to serialize or deserialize it only because this "/cosmos.bank.v1beta1.MsgSend" string is passed along. With this typeUrl, the blockchain client and CosmJS are able to pick the right deserializer. This object is also named MsgSend in cosmjs-types. But in this tutorial, you have picked the deserializer manually.



To learn how to make your own types for your own blockchain project, head to [Create Custom CosmJS Interfaces](https://ida.interchain.io/tutorials/7-cosmjs/5-create-custom.html).

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#deserialize-the-message) Deserialize the message

Now that you know the only message in the transaction is a MsgSend, you need to deserialize it. First add the necessary import at the top:



Copy

import { IndexedTx, StargateClient } from "@cosmjs/stargate"

+ import { MsgSend } from "cosmjs-types/cosmos/bank/v1beta1/tx"

import { Tx } from "cosmjs-types/cosmos/tx/v1beta1/tx"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L2" \t "_blank)

Then you deserialize the message. Add:



Copy

console.log("Decoded messages:", decodedTx.body!.messages)

+ const sendMessage: MsgSend = MsgSend.decode(decodedTx.body!.messages[0].value)

+ console.log("Sent message:", sendMessage)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L2" \t "_blank)

Which, on your next npm run experiment, prints:



Copy

Sent message: {

fromAddress: 'cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he',

toAddress: 'cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf',

amount: [ { denom: 'uatom', amount: '10000000' } ]

}

In this message, the fromAddress is that of the faucet:



Copy

console.log("Sent message:", sendMessage)

+ const faucet: string = sendMessage.fromAddress

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L23" \t "_blank)

Similar to how you got the balance for Alice, you get the faucet's balance as well. Try this by [copying (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts#L24) the code to print Alice's balances. When running, you should get:



Copy

Faucet balances: [ { denom: 'uatom', amount: '867777337235' } ]

****

**Getting the faucet address another way**

Instead of using the decode functions that come with the Tx and MsgSend imports, you process the data yourself via alternative means. If you would like to experiment more, parse the rawLog manually as opposed to deserializing the transaction as suggested previously. Note the conceptual difference between Tx and the rawLog. The Tx, or MsgSend, object is an **input to the computation** that takes place when the transaction is included in a block. The rawLog is the resulting **output** of said computation and its content depends on what the blockchain code emitted when executing the transaction.

In particular, if the transaction failed you would be able to extract the faucet address from the Tx but *not* from the rawLog, because the log would contain the error message.

From the IndexedTx you see that there is a [rawLog (opens new window)](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/stargateclient.ts#L64), which happens to be a stringified JSON.

Copy

const rawLog = JSON.parse(faucetTx.rawLog)

console.log("Raw log:", JSON.stringify(rawLog, null, 4))

experiment.ts

[View source](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L28-L29" \t "_blank)

The structure of the raw log is not always obvious, but in this example it contains:

Copy

[

{

"events": [

...

{

"type": "coin\_spent",

"attributes": [

{

"key": "spender",

"value": "cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"

},

...

]

...

}

...

]

}

]

Because this is a JSON file, you're able to fetch the faucet's address as follows:

Copy

const faucet: string = rawLog[0].events

.find((eventEl: any) => eventEl.type === "coin\_spent")

.attributes.find((attribute: any) => attribute.key === "spender").value

experiment.ts

[View source](https://github.com/b9lab/cosmjs-sandbox/blob/with-stargate-client/experiment.ts" \l "L30-L32" \t "_blank)

Although this is a perfectly valid way to extract specific values from a transaction's message, it is not the recommended way to do so. The benefit of importing the message types is that you do not have to manually dig through the raw log of each Tx you're looking to use in your application.

These actions are example uses of the read-only StargateClient and of the serialization tools that come with CosmJS.



Get the result of the above steps [here (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/tree/with-stargate-client).

Now it is time for Alice to send some tokens back to the faucet.

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#prepare-a-signing-client) Prepare a signing client

If you go through the methods inside [StargateClient (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/stargateclient.ts#L139), you see that it only contains query-type methods and none for facilitating the preparation of transactions. It does have [BroadcastTx (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/stargateclient.ts#L350), a function that can send ready-made transactions.

Now, for Alice to send transactions she needs to be able to sign them. And to be able to sign transactions she needs access to her *private keys* or *mnemonics* (or rather she needs a client that has access to them). That is where [SigningStargateClient (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/signingstargateclient.ts#L147) comes in. Conveniently, SigningStargateClient inherits from StargateClient.

Update your import line:



Copy

- import { IndexedTx, StargateClient } from "@cosmjs/stargate"

+ import { IndexedTx, SigningStargateClient, StargateClient } from "@cosmjs/stargate"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L3" \t "_blank)

Look at its declaration by right-clicking on the SigningStargateClient in your imports and choosing Go to Definition.

When you instantiate SigningStargateClient by using the [connectWithSigner (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/signingstargateclient.ts#L118) method, you need to pass it a [**signer** (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/signingstargateclient.ts#L120). In this case, use the [OfflineDirectSigner (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/proto-signing/src/signer.ts#L21-L24) interface.



The recommended way to encode messages is by using OfflineDirectSigner, which uses Protobuf. However, hardware wallets such as Ledger do not support this and still require the legacy Amino encoder. If your app requires Amino support, you have to use the OfflineAminoSigner.   
  
Read more about encoding [here (opens new window)↗](https://docs.cosmos.network/main/core/encoding.html).

The signer needs access to Alice's **private key**, and there are several ways to accomplish this. In this example, use Alice's saved **mnemonic**. To load the mnemonic as text in your code you need this import:



Copy

+ import { readFile } from "fs/promises"

import { IndexedTx, SigningStargateClient, StargateClient } from "@cosmjs/stargate"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L1" \t "_blank)

There are several implementations of OfflineDirectSigner available. The [DirectSecp256k1HdWallet (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/proto-signing/src/directsecp256k1hdwallet.ts#L133) implementation is most relevant to us due to its [fromMnemonic (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/proto-signing/src/directsecp256k1hdwallet.ts#L140-L141) method. Add the import:



Copy

import { readFile } from "fs/promises"

+ import { DirectSecp256k1HdWallet, OfflineDirectSigner } from "@cosmjs/proto-signing"

import { IndexedTx, SigningStargateClient, StargateClient } from "@cosmjs/stargate"

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L2" \t "_blank)

The fromMnemonic factory function needs a string with the mnemonic. You read this string from the mnemonic file. Create a new top-level function that returns an OfflineDirectSigner:



Copy

const getAliceSignerFromMnemonic = async (): Promise<OfflineDirectSigner> => {

return DirectSecp256k1HdWallet.fromMnemonic((await readFile("./testnet.alice.mnemonic.key")).toString(), {

prefix: "cosmos",

})

}

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L9-L13" \t "_blank)

The Cosmos Hub Testnet uses the cosmos address prefix. This is the default used by DirectSecp256k1HdWallet, but you are encouraged to explicitly define it as you might be working with different prefixes on different blockchains. Lower down in your runAll function, add:



Copy

const aliceSigner: OfflineDirectSigner = await getAliceSignerFromMnemonic()

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L44" \t "_blank)

As a first step, confirm that it recovers Alice's address as expected:



Copy

const aliceSigner: OfflineDirectSigner = await getAliceSignerFromMnemonic()

+ const alice = (await aliceSigner.getAccounts())[0].address

+ console.log("Alice's address from signer", alice)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L45-L46" \t "_blank)

Now add the line that finally creates the signing client:



Copy

console.log("Alice's address from signer", alice)

+ const signingClient = await SigningStargateClient.connectWithSigner(rpc, aliceSigner)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L47" \t "_blank)

Check that it works like the read-only client that you used earlier, and from which [it inherits (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/signingstargateclient.ts#L109), by adding:



Copy

const signingClient = await SigningStargateClient.connectWithSigner(rpc, aliceSigner)

+ console.log(

+ "With signing client, chain id:",

+ await signingClient.getChainId(),

+ ", height:",

+ await signingClient.getHeight()

+ )

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/with-signing-stargate-client/experiment.ts" \l "L48-L53" \t "_blank)

Run it with npm run experiment.



Get the result of the previous steps [here (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/tree/with-signing-stargate-client).

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#send-tokens) Send tokens

Alice can now send some tokens back to the faucet, but to do so she also needs to pay the network's gas fee. How much gas should she use, and at what price?

She can copy what the faucet did. To discover this, add:



Copy

console.log(

...

await signingClient.getHeight()

)

+ console.log("Gas fee:", decodedTx.authInfo!.fee!.amount)

+ console.log("Gas limit:", decodedTx.authInfo!.fee!.gasLimit.toString(10))

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens/experiment.ts" \l "L55-L56" \t "_blank)

When you run it, it prints:



Copy

Gas fee: [ { denom: 'uatom', amount: '500' } ]

Gas limit: 200000

With the gas information now decided, how does Alice structure her command so that she sends 1% of her holdings, i.e. 100000uatom, back to the faucet? SigningStargateClient's [sendTokens (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/stargate/src/signingstargateclient.ts#L180-L186) function takes a Coin[] as input. Coin is simply defined as:



Copy

export interface Coin {

denom: string;

amount: string;

}

src /

cosmos /

... /

v1beta1 /

coin.ts

[View source→](https://github.com/confio/cosmjs-types/blob/v0.4.1/src/cosmos/base/v1beta1/coin.ts" \l "L13-L16" \t "_blank)

Alice can pick any denom and any amount as long as she owns them, the signing client signs the transaction and broadcasts it. In this case it is:



Copy

{ denom: "uatom", amount: "100000" }

With this gas and coin information, add the command:



Copy

console.log("Gas limit:", decodedTx.authInfo!.fee!.gasLimit.toString(10))

+ // Check the balance of Alice and the Faucet

+ console.log("Alice balance before:", await client.getAllBalances(alice))

+ console.log("Faucet balance before:", await client.getAllBalances(faucet))

+ // Execute the sendTokens Tx and store the result

+ const result = await signingClient.sendTokens(

+ alice,

+ faucet,

+ [{ denom: "uatom", amount: "100000" }],

+ {

+ amount: [{ denom: "uatom", amount: "500" }],

+ gas: "200000",

+ },

+ )

+ // Output the result of the Tx

+ console.log("Transfer result:", result)

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens/experiment.ts" \l "L57-L63" \t "_blank)

To confirm that it worked, add another balance check:



Copy

console.log("Transfer result:", result)

+ console.log("Alice balance after:", await client.getAllBalances(alice))

+ console.log("Faucet balance after:", await client.getAllBalances(faucet))

experiment.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens/experiment.ts" \l "L64-L65" \t "_blank)

Run this with npm run experiment and you should get:



Copy

...

Transfer result: {

code: 0,

height: 9507151,

rawLog: '[{"events":[{"type":"coin\_received","attributes":[{"key":"receiver","value":"cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"},{"key":"amount","value":"100000uatom"}]},{"type":"coin\_spent","attributes":[{"key":"spender","value":"cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf"},{"key":"amount","value":"100000uatom"}]},{"type":"message","attributes":[{"key":"action","value":"/cosmos.bank.v1beta1.MsgSend"},{"key":"sender","value":"cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf"},{"key":"module","value":"bank"}]},{"type":"transfer","attributes":[{"key":"recipient","value":"cosmos15aptdqmm7ddgtcrjvc5hs988rlrkze40l4q0he"},{"key":"sender","value":"cosmos17tvd4hcszq7lcxuwzrqkepuau9fye3dal606zf"},{"key":"amount","value":"100000uatom"}]}]}]',

transactionHash: '7F770F24CB3C805FE45A8D26DD5EC5AA3F7B906AA7D6CB1F3FE8B554CBA93E12',

gasUsed: 74190,

gasWanted: 200000

}

Alice balance after: [ { denom: 'uatom', amount: '9899500' } ]

Faucet balance after: [ { denom: 'uatom', amount: '867777437235' } ]

According to the rawLog, the faucet received 100000uatom. Since Alice ends up with "9899500uatom", it means she also paid 500uatom for gas.

This concludes your first use of CosmJS to send tokens.



Find the result of all the previous steps [here (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/tree/send-tokens).



You connected to a publicly running testnet. Therefore, you depended on someone else to have a blockchain running with an open and publicly available RPC port and faucet. What if you wanted to try connecting to your own locally running blockchain?

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#with-a-locally-started-chain) With a locally started chain

The easiest option is to reuse the simd chain that you started in [another tutorial](https://ida.interchain.io/tutorials/3-run-node/). Make sure that you have created two accounts, Alice and Bob. You also sent tokens using simd. Be sure to credit enough tokens to Alice.

When you finally launch simd:

**Local**

**WSL2**



Copy

$ ./build/simd start

Copy

$ ./build/simd start --keyring-backend test

You see the line:



Copy

...

4:37PM INF Starting RPC HTTP server on 127.0.0.1:26657 module=rpc-server

...

Port 26657 is the default port for RPC endpoints built with the SDK, unless otherwise configured in ~/.simapp/config/config.toml. 127.0.0.1:26657 is the URL you need to add to your script later.

Make a copy of your experiment.ts script, with some adjustments. Name it experiment-local.ts. Add a new run target in package.json:



Copy

{

...

"scripts": {

"experiment": "ts-node experiment.ts",

+ "experiment-local": "ts-node experiment-local.ts",

...

}

}

package.json

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/package.json" \l "L8" \t "_blank)

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#preparing-your-keys) Preparing your keys

Although you have Alice's address, you may not have her mnemonic or private key. The private key is stored in your operating system's keyring backend. For the purpose of this exercise, extract it - generally this is an unsafe operation:

**Local**

**WSL2**



Copy

$ ./build/simd keys export alice --unsafe --unarmored-hex

Copy

$ ./build/simd keys export alice --unsafe --unarmored-hex \

--keyring-backend test

You get a 64-digit-long hex value. Copy-paste it into a new simd.alice.private.key file in your cosmjs-sandbox folder. The .gitignore was already configured earlier to ignore it, which mitigates the risk.



If you cannot remember which alias you gave your keys, list them:

**Local**

**WSL2**



Copy

$ ./build/simd keys list

Copy

$ ./build/simd keys list --keyring-backend test

Which returns:



Copy

- address: cosmos1c3srguwnzah5nd4cn49shltvr6tsrcl2jwn8je

name: alice

pubkey: '{"@type":"/cosmos.crypto.secp256k1.PubKey","key":"AhR7SWWDsaSxBD9r/mIhbVOWap70jA3WpBIqjOJo4Dwp"}'

type: local

- address: cosmos1umpxwaezmad426nt7dx3xzv5u0u7wjc0kj7ple

name: bob

pubkey: '{"@type":"/cosmos.crypto.secp256k1.PubKey","key":"Av1VW23/laXWtbwWwOUHCvjjeLLqbdzazRneeRsE/shL"}'

type: local

[#Copy link](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#update-your-script) Update your script

With the new elements in place, update your experiment-local.ts script. Change rpc:



Copy

- const rpc = "https://rpc.sentry-01.theta-testnet.polypore.xyz"

+ const rpc = "http://127.0.0.1:26657"

experiment-local.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts" \l "L6" \t "_blank)

And skip the lengthy process to get the faucet address. Just set faucet to Bob's address:



Copy

- const faucetTx: IndexedTx = (await client.getTx(

- "540484BDD342702F196F84C2FD42D63FA77F74B26A8D7383FAA5AB46E4114A9B"

- ))!

- console.log("Faucet Tx:", faucetTx)

- const decodedTx: Tx = Tx.decode(faucetTx.tx)

- console.log("DecodedTx:", decodedTx)

- console.log("Decoded messages:", decodedTx.body!.messages)

- const sendMessage: MsgSend = MsgSend.decode(decodedTx.body!.messages[0].value)

- console.log("Sent message:", sendMessage)

- const faucet: string = sendMessage.fromAddress

+ const faucet: string = "cosmos1umpxwaezmad426nt7dx3xzv5u0u7wjc0kj7ple"

experiment-local.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts" \l "L22" \t "_blank)

Next, you need to replace the function to create Alice's signer because you're using a private key instead of a mnemonic, so the fromMnemonic method that comes with DirectSecp256k1HdWallet does not work. The [fromKey (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.2/packages/proto-signing/src/directsecp256k1wallet.ts#L21) method that comes with DirectSecp256k1Wallet is the more appropriate choice this time.

Adjust the import:



Copy

- import { DirectSecp256k1HdWallet, OfflineDirectSigner } from "@cosmjs/proto-signing"

+ import { DirectSecp256k1Wallet, OfflineDirectSigner } from "@cosmjs/proto-signing"

experiment-local.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts" \l "L3" \t "_blank)

In DirectSecp256k1Wallet the fromKey factory function needs a Uint8Array. Fortunately, CosmJS includes a utility to convert a hexadecimal string into a Uint8Array. Import it:



Copy

import { readFile } from "fs/promises"

+ import { fromHex } from "@cosmjs/encoding"

import { DirectSecp256k1Wallet, OfflineDirectSigner } from "@cosmjs/proto-signing"

experiment-local.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts" \l "L2" \t "_blank)

Now create a new function to get a signer to replace the previous one:



Copy

const getAliceSignerFromPriKey = async(): Promise<OfflineDirectSigner> => {

return DirectSecp256k1Wallet.fromKey(

fromHex((await readFile("./simd.alice.private.key")).toString()),

"cosmos",

)

}

experiment-local.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts" \l "L8-L13" \t "_blank)

Replace getAliceSignerFromMnemonic with the newly created getAliceSignerFromPriKey:



Copy

const faucet: string = "cosmos1umpxwaezmad426nt7dx3xzv5u0u7wjc0kj7ple"

- const aliceSigner: OfflineDirectSigner = await getAliceSignerFromMnemonic()

+ const aliceSigner: OfflineDirectSigner = await getAliceSignerFromPriKey()

experiment-local.ts

[View source→](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts" \l "L24" \t "_blank)

Also change the token unit from uatom [to stake (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/blob/send-tokens-local/experiment-local.ts#L37-L38) in your sendTokens transaction, because this is the default token when using simapp. Experiment with adjusting the values as desired. Run it with:

**Local**

**Docker**



Copy

$ npm run experiment-local

Copy

$ docker run --rm -it \

-v $(pwd):/sandbox -w /sandbox \

node:lts-slim \

npm run experiment-local

And confirm the output is as expected. For instance something like:



Copy

> cosmjs-sandbox@1.0.0 experiment-local

> ts-node experiment-local.ts

With client, chain id: demo , height: 44883

Alice balances: []

Alice's address from signer cosmos1c3srguwnzah5nd4cn49shltvr6tsrcl2jwn8je

With signing client, chain id: demo , height: 44883

Alice balance before: [ { denom: 'stake', amount: '19891835' } ]

Faucet balance before: [ { denom: 'stake', amount: '10010000' } ]

Transfer result: {

code: 0,

height: 44885,

rawLog: '[{"events":[{"type":"coin\_received","attributes":[{"key":"receiver","value":"cosmos1umpxwaezmad426nt7dx3xzv5u0u7wjc0kj7ple"},{"key":"amount","value":"100000stake"}]},{"type":"coin\_spent","attributes":[{"key":"spender","value":"cosmos1c3srguwnzah5nd4cn49shltvr6tsrcl2jwn8je"},{"key":"amount","value":"100000stake"}]},{"type":"message","attributes":[{"key":"action","value":"/cosmos.bank.v1beta1.MsgSend"},{"key":"sender","value":"cosmos1c3srguwnzah5nd4cn49shltvr6tsrcl2jwn8je"},{"key":"module","value":"bank"}]},{"type":"transfer","attributes":[{"key":"recipient","value":"cosmos1umpxwaezmad426nt7dx3xzv5u0u7wjc0kj7ple"},{"key":"sender","value":"cosmos1c3srguwnzah5nd4cn49shltvr6tsrcl2jwn8je"},{"key":"amount","value":"100000stake"}]}]}]',

transactionHash: 'A49EBD41E37CDACF258F0BCD0954C52138FB5121C9A3B58138A2279EDB526B6D',

gasUsed: 72702,

gasWanted: 200000

}

Alice balance after: [ { denom: 'stake', amount: '19791335' } ]

Faucet balance after: [ { denom: 'stake', amount: '10110000' } ]

You have now used CosmJS's bank module on a locally running Cosmos blockchain.



Find the complete set of files [here (opens new window)↗](https://github.com/b9lab/cosmjs-sandbox/tree/send-tokens-local).



If you want a detailed introduction to CosmJS, check out the following video with Noam Cohan, former Developer Relations Lead for Cosmos Hub.



If you would like to see how to do more actions when listening to events from your own checkers game, you can go straight to the related exercise in [CosmJS for Your Chain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/).

More specifically, you can jump to:

* [Backend Scripts for Game Indexing](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/5-server-side.html), for a Web2.0 server listening to game events.

synopsis

To summarize, this section has explored:

* How to gain familiarity with CosmJS by implementing a basic feature of the Interchain Ecosystem, the ability to send tokens via the bank module.
* How to clone a ready-made test repository and install the required modules in order to experiment with CosmJS, for which NodeJs and Visual Studio Code will be required.
* How to connect to a public node in the Interchain Ecosystem, acquire a wallet address on a testnet, and create a key for an imaginary user for the purposes of experimenting.
* How to add your imports, define your connection, get a balance, get the faucet address, prepare a signing client, and successfully send tokens on a chain being run by someone else.
* How to connect with your own locally running blockchain, including how to prepare your keys and update your script.

previous

[](https://ida.interchain.io/tutorials/7-cosmjs/1-cosmjs-intro.html)

**[What is CosmJS?](https://ida.interchain.io/tutorials/7-cosmjs/1-cosmjs-intro.html)**

up next

**[Compose Complex Transactions](https://ida.interchain.io/tutorials/7-cosmjs/3-multi-msg.html)**

[[](https://ida.interchain.io/tutorials/7-cosmjs/3-multi-msg.html)](https://ida.interchain.io/tutorials/7-cosmjs/3-multi-msg.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

[](https://ida.interchain.io/ida-course/discord-info.html)

On this page

[Script preparation](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#script-preparation)

[Testnet preparation](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#testnet-preparation)

[Add your imports](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#add-your-imports)

[Define your connection](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#define-your-connection)

[Get a balance](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#get-a-balance)

[Get the faucet address](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#get-the-faucet-address)

[Deserialize the transaction](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#deserialize-the-transaction)

[What is this long string?](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#what-is-this-long-string)

[Deserialize the message](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#deserialize-the-message)

[Prepare a signing client](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#prepare-a-signing-client)

[Send tokens](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#send-tokens)

[With a locally started chain](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#with-a-locally-started-chain)

[Preparing your keys](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#preparing-your-keys)

[Update your script](https://ida.interchain.io/tutorials/7-cosmjs/2-first-steps.html#update-your-script)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)