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

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 0 - Getting Started](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Getting Started](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Blockchain 101](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Blockchain History](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Public and Managed Blockchains](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Consensus in Distributed Networks](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Cryptography](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Self-Assessment Quiz](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Go Introduction - First Steps](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Go Basics](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Go Interfaces](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Control Structures in Go](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Arrays and Slices in Go](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Standard Packages in Go](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Concurrency in Go](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Docker Introduction](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[The Interchain Ecosystem](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Getting ATOM and Staking It](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[A Blockchain App Architecture](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Accounts](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Transactions](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Modules](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Protobuf](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Multistore and Keepers](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[BaseApp](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Queries](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Events](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Context](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Testing](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Relaying with IBC](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Interchain Security](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Bridges](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Migrations](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 1 Quiz](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 2 - First Steps](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[First Steps](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Setup Your Work Environment](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Run a Node, API, and CLI](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Ignite CLI](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Store Object](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create and Save a Game Properly](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Add a Way to Make a Move](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Emit Game Information](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Record the Game Winner](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 2 Exercise](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[What is IBC?](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC Token Transfer](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC Tooling](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[What is CosmJS?](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Your First CosmJS Actions](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Compose Complex Transactions](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Learn to Integrate Keplr](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Put Your Games in Order](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Auto-Expiring Games](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Let Players Set a Wager](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Handle wager payments](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Integration tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Incentivize Players](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Help Find a Correct Move](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Understand IBC Denoms](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Go Relayer](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Hermes Relayer](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create Custom Objects](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Get an External GUI](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Backend Script for Game Indexing](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[IBC Application Developer Introduction](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Create a Leaderboard Chain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Run in Production](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Prepare the Software to Run](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Prepare a Validator and Keys](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Prepare Where the Node Starts](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Prepare and Do Migrations](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Simulate Production in Docker](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Tally Player Info After Production](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Simulate a Migration in Docker](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Final Exam](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[What's Next?](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

[Continue Your Interchain Journey](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html)

Docs Version Switcher

On this page

[Encodable messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#encodable-messages)

[A signing Stargate for checkers](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#a-signing-stargate-for-checkers)

[The action methods](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#the-action-methods)

[Integration tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#integration-tests)

[Key preparation](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#key-preparation)

[Token preparation](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#token-preparation)

[Adding tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#adding-tests)

[Create and play](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#create-and-play)

[Multiple transactions in a block](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#multiple-transactions-in-a-block)

[Combine messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#combine-messages)

[Further tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#further-tests)

[Prepare your checkers chain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#prepare-your-checkers-chain)

[Launch the tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#launch-the-tests)

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#create-custom-messages) **Create Custom Messages**



Make sure you have all you need before proceeding:

* You understand the concepts of [CosmJS](https://ida.interchain.io/tutorials/7-cosmjs/1-cosmjs-intro.html).
* You have generated the necessary TypeScript types in [the previous tutorial](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/1-cosmjs-objects.html). If not, just clone and checkout the [relevant branch (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/tree/stargate).

In the previous section, you created the objects that allow you to **query** your checkers blockchain. Now, you will create the elements that allow you to **send transactions** to it.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#encodable-messages) Encodable messages

Previously you defined in Protobuf two messages and their respective responses. You had Protobuf [compile them (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/generated/src/types/generated/checkers/tx.ts). Now you will create a few instances of EncodeObject, similar to how this is done in CosmJS's [bank module (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/modules/bank/messages.ts). First, collect their names and Protobuf packages. Each Protobuf type identifier is assigned its encodable type:



Copy

export const typeUrlMsgCreateGame = "/alice.checkers.checkers.MsgCreateGame"

export const typeUrlMsgCreateGameResponse = "/alice.checkers.checkers.MsgCreateGameResponse"

export const typeUrlMsgPlayMove = "/alice.checkers.checkers.MsgPlayMove"

export const typeUrlMsgPlayMoveResponse = "/alice.checkers.checkers.MsgPlayMoveResponse"

export const checkersTypes: ReadonlyArray<[string, GeneratedType]> = [

[typeUrlMsgCreateGame, MsgCreateGame],

[typeUrlMsgCreateGameResponse, MsgCreateGameResponse],

[typeUrlMsgPlayMove, MsgPlayMove],

[typeUrlMsgPlayMoveResponse, MsgPlayMoveResponse],

]

src /

types /

checkers /

messages.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/messages.ts" \l "L11-L21" \t "_blank)

Next proceed with the declarations. As with CosmJS, you can add an isMsgXX helper for each:



Copy

export interface MsgCreateGameEncodeObject extends EncodeObject {

readonly typeUrl: "/alice.checkers.checkers.MsgCreateGame"

readonly value: Partial<MsgCreateGame>

}

export function isMsgCreateGameEncodeObject(

encodeObject: EncodeObject,

): encodeObject is MsgCreateGameEncodeObject {

return (

(encodeObject as MsgCreateGameEncodeObject).typeUrl === typeUrlMsgCreateGame

)

}

export interface MsgCreateGameResponseEncodeObject... {}

export function isMsgCreateGameResponseEncodeObject(...)...

export interface MsgPlayMoveEncodeObject... {}

...

src /

types /

checkers /

messages.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/messages.ts" \l "L23-L43" \t "_blank)

This needs to be repeated for each of the messages that you require. To refresh your memory, that is:

* [MsgCreateGame (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/messages.ts#L24-L32)
* [MsgCreateGameResponse (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/messages.ts#L34-L43)
* [MsgPlayMove (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/messages.ts#L45-L54)
* [MsgPlayMoveResponse (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/messages.ts#L56-L65)

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#a-signing-stargate-for-checkers) A signing Stargate for checkers

This process again takes inspiration from [SigningStargateClient (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/signingstargateclient.ts#L55-L69). Prepare by registering your new types in addition to the others, so that your client knows them all:



Copy

import { defaultRegistryTypes } from "@cosmjs/stargate"

export const checkersDefaultRegistryTypes: ReadonlyArray<[string, GeneratedType]> = [

...defaultRegistryTypes,

...checkersTypes,

]

function createDefaultRegistry(): Registry {

return new Registry(checkersDefaultRegistryTypes)

}

src /

checkers\_signingstargateclient.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/checkers_signingstargateclient.ts" \l "L23-L30" \t "_blank)

Similar to the read-only CheckersStargateClient, create a CheckersSigningStargateClient that inherits from SigningStargateClient:



Copy

export class CheckersSigningStargateClient extends SigningStargateClient {

public readonly checkersQueryClient: CheckersExtension | undefined

public static async connectWithSigner(

endpoint: string,

signer: OfflineSigner,

options: SigningStargateClientOptions = {},

): Promise<CheckersSigningStargateClient> {

const tmClient = await Tendermint34Client.connect(endpoint)

return new CheckersSigningStargateClient(tmClient, signer, {

registry: createDefaultRegistry(),

...options,

})

}

protected constructor(

tmClient: Tendermint34Client | undefined,

signer: OfflineSigner,

options: SigningStargateClientOptions,

) {

super(tmClient, signer, options)

if (tmClient) {

this.checkersQueryClient = QueryClient.withExtensions(tmClient, setupCheckersExtension)

}

}

}

src /

checkers\_signingstargateclient.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/checkers_signingstargateclient.ts" \l "L32-L56" \t "_blank)

Note the use of createDefaultRegistry as the default registry, if nothing was passed via the options.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#the-action-methods) The action methods

Finally you ought to add the methods that allow you to interact with the blockchain. They also help advertise how to craft messages for your client. Taking inspiration from [sendTokens (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/signingstargateclient.ts#L180-L197), create one function for each of your messages:



Copy

public async createGame(

creator: string,

black: string,

red: string,

denom: string,

wager: Long,

fee: StdFee | "auto" | number,

memo = "",

): Promise<DeliverTxResponse> {

const createMsg: MsgCreateGameEncodeObject = {

typeUrl: typeUrlMsgCreateGame,

value: {

black: black,

red: red,

creator: creator,

denom: denom,

wager: wager,

},

}

return this.signAndBroadcast(creator, [createMsg], fee, memo)

}

public async playMove(

creator: string,

gameIndex: string,

from: Pos,

to: Pos,

fee: StdFee | "auto" | number,

memo = "",

): Promise<DeliverTxResponse> {

const playMoveMsg: MsgPlayMoveEncodeObject = {

typeUrl: typeUrlMsgPlayMove,

value: {

creator: creator,

gameIndex: gameIndex,

fromX: Long.fromNumber(from.x),

fromY: Long.fromNumber(from.y),

toX: Long.fromNumber(to.x),

toY: Long.fromNumber(to.y),

},

}

return this.signAndBroadcast(creator, [playMoveMsg], fee, memo)

}

src /

checkers\_signingstargateclient.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/checkers_signingstargateclient.ts" \l "L58-L100" \t "_blank)



You should not consider these two functions as the **only** ones that users of CheckersSigningStargateClient should ever use. Rather, see them as a demonstration regarding to how to build messages properly.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#integration-tests) Integration tests

You can reuse the setup you prepared in the previous section. However, there is an added difficulty: because you send transactions, your tests need access to keys and tokens. How do you provide them in a testing context?

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#key-preparation) Key preparation

You would **not** treat mainnet keys in this way, but here you save testing keys on disk. Update .env with the test mnemonics of your choice:

**Local**

**Docker**



Copy

RPC\_URL="http://localhost:26657"

+ MNEMONIC\_TEST\_ALICE="theory arrow blue much illness carpet arena thought clay office path museum idea text foot bacon until tragic inform stairs pitch danger spatial slight"

+ ADDRESS\_TEST\_ALICE="cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql"

+ MNEMONIC\_TEST\_BOB="apple spoil melody venture speed like dawn cherry insane produce carry robust duck language next electric episode clinic acid sheriff video knee spoil multiply"

+ ADDRESS\_TEST\_BOB="cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22"

.env

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/.env" \l "L3-L6" \t "_blank)

Copy

RPC\_URL="http://checkers:26657"

+ MNEMONIC\_TEST\_ALICE="theory arrow blue much illness carpet arena thought clay office path museum idea text foot bacon until tragic inform stairs pitch danger spatial slight"

+ ADDRESS\_TEST\_ALICE="cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql"

+ MNEMONIC\_TEST\_BOB="apple spoil melody venture speed like dawn cherry insane produce carry robust duck language next electric episode clinic acid sheriff video knee spoil multiply"

+ ADDRESS\_TEST\_BOB="cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22"

.env

[View source](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/.env" \l "L3-L6" \t "_blank)

If you use different mnemonics and do not yet know the corresponding addresses, you can get them from the before action (below) when it fails. Also adjust environment.d.ts to inform the TypeScript compiler:



Copy

declare global {

namespace NodeJS {

interface ProcessEnv {

...

+ MNEMONIC\_TEST\_ALICE: string

+ ADDRESS\_TEST\_ALICE: string

+ MNEMONIC\_TEST\_BOB: string

+ ADDRESS\_TEST\_BOB: string

}

}

}

...

environment.d.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/environment.d.ts" \l "L6-L9" \t "_blank)

In a new separate file, define how you build a signer from the mnemonic:



Copy

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

export const getSignerFromMnemonic = async (mnemonic: string): Promise<OfflineDirectSigner> => {

return DirectSecp256k1HdWallet.fromMnemonic(mnemonic, {

prefix: "cosmos",

})

}

src /

util /

signer.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/util/signer.ts" \l "L1-L8" \t "_blank)

Create a new stored-game-action.ts integration test file, modeled on stored-game.ts, that starts by preparing the signers and confirms the match between mnemonics and first addresses:



Copy

const { RPC\_URL, ADDRESS\_TEST\_ALICE: alice, ADDRESS\_TEST\_BOB: bob } = process.env

let aliceSigner: OfflineDirectSigner, bobSigner: OfflineDirectSigner

before("create signers", async function () {

aliceSigner = await getSignerFromMnemonic(process.env.MNEMONIC\_TEST\_ALICE)

bobSigner = await getSignerFromMnemonic(process.env.MNEMONIC\_TEST\_BOB)

expect((await aliceSigner.getAccounts())[0].address).to.equal(alice)

expect((await bobSigner.getAccounts())[0].address).to.equal(bob)

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L30-L38" \t "_blank)

These early verifications do not need any running chain, so go ahead and make sure they pass. Add a temporary empty it test and run:

**Local**

**Docker**



Copy

$ npm test

Copy

$ docker run --rm -it \

-v $(pwd):/client \

-w /client \

node:18.7-slim \

npm test

With this confirmation, you can add another before that creates the signing clients:



Copy

let aliceClient: CheckersSigningStargateClient,

bobClient: CheckersSigningStargateClient,

checkers: CheckersExtension["checkers"]

before("create signing clients", async function () {

aliceClient = await CheckersSigningStargateClient.connectWithSigner(RPC\_URL, aliceSigner, {

gasPrice: GasPrice.fromString("0stake"),

})

bobClient = await CheckersSigningStargateClient.connectWithSigner(RPC\_URL, bobSigner, {

gasPrice: GasPrice.fromString("0stake"),

})

checkers = aliceClient.checkersQueryClient!.checkers

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L40-L52" \t "_blank)

If the running chain allows it, and to make your life easier, you can set the gas price to 0. If not, set it as low as possible.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#token-preparation) Token preparation

Just saving keys on disk does not magically make these keys hold tokens on your test blockchain. You need to fund them at their addresses, using the funds of other addresses of your running chain. A faucet, if one is set up, is convenient for this step.

1. If you use Ignite, it has created a faucet endpoint for you at port 4500. The page http://localhost:4500 explains the API.
2. If you use the [CosmJS faucet (opens new window)↗](https://www.npmjs.com/package/@cosmjs/faucet), you can make it serve on port 4500 too.

For the purpose of this exercise, you will prepare for both faucets.

Add the faucet(s) address in .env:

**Local**

**Docker**



Copy

RPC\_URL="http://localhost:26657"

+ FAUCET\_URL="http://localhost:4500"

...

.env

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/.env" \l "L2" \t "_blank)

Copy

RPC\_URL="http://checkers:26657"

+ FAUCET\_URL="http://cosmos-faucet:4500"

...

.env

[View source](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/.env" \l "L2" \t "_blank)

The name cosmos-faucet will be used when starting the faucet container.

Also add the faucet address to environment.d.ts:



Copy

declare global {

namespace NodeJS {

interface ProcessEnv {

RPC\_URL: string

+ FAUCET\_URL: string

...

}

}

}

...

environment.d.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/environment.d.ts" \l "L5" \t "_blank)

In a new, separate file, add:

1. A http helper function:



Copy

export const httpRequest = async (

url: string | URL,

options: RequestOptions,

postData: string,

): Promise<string> =>

new Promise((resolve, reject) => {

let all = ""

const req = http.request(url, options, (response: IncomingMessage) => {

response.setEncoding("utf8")

response.on("error", reject)

response.on("end", () => {

if (400 <= response.statusCode!) reject(all)

else resolve(all)

})

response.on("data", (chunk) => (all = all + chunk))

})

req.write(postData)

req.end()

})

src /

util /

faucet.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/util/faucet.ts" \l "L4-L22" \t "_blank)

1. Two helper functions to easily call the faucets of [CosmJS (opens new window)↗](https://www.npmjs.com/package/@cosmjs/faucet) and [Ignite](https://ida.interchain.io/hands-on-exercise/1-ignite-cli/1-ignitecli.html#your-gui):

**for CosmJS**

**for Ignite**



Copy

export const askFaucetComsJs = async (

address: string,

tokens: { [denom: string]: number },

): Promise<string[]> =>

Promise.all(

Object.keys(tokens).map((denom) =>

httpRequest(

`${process.env.FAUCET\_URL}/credit`,

{

method: "POST",

headers: { "Content-Type": "application/json" },

},

JSON.stringify({ address, denom }),

),

),

)

src /

util /

faucet.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/util/faucet.ts" \l "L30-L45" \t "_blank)

Copy

export const askFaucetIgniteCli = async (

address: string,

tokens: { [denom: string]: number },

): Promise<string> =>

httpRequest(

process.env.FAUCET\_URL,

{ method: "POST" },

JSON.stringify({

address: address,

coins: Object.entries(tokens).map(([denom, value]) => value + denom),

}),

)

src

util

faucet.ts

[View source](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/util/faucet.ts" \l "L47-L58" \t "_blank)



* + The CosmJS faucet API **does not** let you specify the desired amount.
  + The Ignite faucet API **does** let you specify the desired amount.

1. A function that tries one faucet, and if that one fails tries the other:



Copy

export const askFaucet = async (

address: string,

tokens: { [denom: string]: number },

): Promise<string | string[]> =>

askFaucetComsJs(address, tokens).catch(() => askFaucetIgniteCli(address, tokens))

src /

util /

faucet.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/util/faucet.ts" \l "L24-L28" \t "_blank)

You will find out with practice how many tokens your accounts need for their tests. Start with any value. Ignite's default configuration is to start a chain with two tokens: stake and token. This is a good default, as you can use both denoms.

Create another before that will credit Alice and Bob from the faucet so that they are rich enough to continue:



Copy

const aliceCredit = {

stake: 100,

token: 1,

},

bobCredit = {

stake: 100,

token: 1,

}

before("credit test accounts", async function () {

this.timeout(40\_000)

if (

parseInt((await aliceClient.getBalance(alice, "stake")).amount, 10) < aliceCredit.stake ||

parseInt((await aliceClient.getBalance(alice, "token")).amount, 10) < aliceCredit.token

)

await askFaucet(alice, aliceCredit)

expect(parseInt((await aliceClient.getBalance(alice, "stake")).amount, 10)).to.be.greaterThanOrEqual(

aliceCredit.stake,

)

expect(parseInt((await aliceClient.getBalance(alice, "token")).amount, 10)).to.be.greaterThanOrEqual(

aliceCredit.token,

)

if (

parseInt((await bobClient.getBalance(bob, "stake")).amount, 10) < bobCredit.stake ||

parseInt((await bobClient.getBalance(bob, "token")).amount, 10) < bobCredit.token

)

await askFaucet(bob, bobCredit)

expect(parseInt((await bobClient.getBalance(bob, "stake")).amount, 10)).to.be.greaterThanOrEqual(

bobCredit.stake,

)

expect(parseInt((await bobClient.getBalance(bob, "token")).amount, 10)).to.be.greaterThanOrEqual(

bobCredit.token,

)

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L54-L87" \t "_blank)

Your accounts are now ready to proceed with the tests proper.



There are an extra 40 seconds given for this potentially slower process: this.timeout(40\_000).

You may want to adjust this time-out value. Here it is set at *10 seconds multiplied by the maximum number of transactions in the function*. Here there are at most 4 transactions when calling the CosmJS faucet. Query calls are typically very fast, and therefore need not enter in the time-out calculation.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#adding-tests) Adding tests

Since these integration tests make calls to a running chain, they need to run one after the other. And if one it fails, all the it tests that come after will fail too. This is not ideal but is how these examples will work.

With a view to reusing them, add convenience methods that encapsulate the extraction of information from the events:

* To get the id of the game created:



Copy

export type GameCreatedEvent = Event

export const getCreateGameEvent = (log: Log): GameCreatedEvent | undefined =>

log.events?.find((event: Event) => event.type === "new-game-created")

export const getCreatedGameId = (createdGameEvent: GameCreatedEvent): string =>

createdGameEvent.attributes.find((attribute: Attribute) => attribute.key == "game-index")!.value

src /

types /

checkers /

events.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/events.ts" \l "L4-L10" \t "_blank)

* To get the captured position and the winner if they exist:



Copy

export type MovePlayedEvent = Event

export const getMovePlayedEvent = (log: Log): MovePlayedEvent | undefined =>

log.events?.find((event: Event) => event.type === "move-played")

export const getCapturedPos = (movePlayedEvent: MovePlayedEvent): Pos | undefined => {

const x: number = parseInt(

movePlayedEvent.attributes.find((attribute: Attribute) => attribute.key == "captured-x")!.value,

10,

)

const y = parseInt(

movePlayedEvent.attributes.find((attribute: Attribute) => attribute.key == "captured-y")!.value,

10,

)

if (isNaN(x) || isNaN(y)) return undefined

return { x, y }

}

export const getWinner = (movePlayedEvent: MovePlayedEvent): GamePiece | undefined =>

movePlayedEvent.attributes.find((attribute: Attribute) => attribute.key == "winner")!.value as

| GamePiece

| undefined

src /

types /

checkers /

events.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/events.ts" \l "L12-L33" \t "_blank)

Here you also define GamePiece as:



Copy

export type GamePiece = Player | "\*"

src /

types /

checkers /

player.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/player.ts" \l "L2" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#create-and-play) Create and play

Start by creating a game, extracting its index from the logs, and confirming that you can fetch it.



Copy

let gameIndex: string

it("can create game with wager", async function () {

this.timeout(10\_000)

const response: DeliverTxResponse = await aliceClient.createGame(

alice,

alice,

bob,

"token",

Long.fromNumber(1),

"auto",

)

const logs: Log[] = JSON.parse(response.rawLog!)

expect(logs).to.be.length(1)

gameId = getCreatedGameId(getCreateGameEvent(logs[0])!)

const game: StoredGame = (await checkers.getStoredGame(gameId))!

expect(game).to.include({

index: gameId,

black: alice,

red: bob,

denom: "token",

})

expect(game.wager.toNumber()).to.equal(1)

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L91-L112" \t "_blank)

Next, add a test that confirms that the wager tokens are consumed on first play:



Copy

it("can play first moves and pay wager", async function () {

this.timeout(20\_000)

const aliceBalBefore = parseInt((await aliceClient.getBalance(alice, "token")).amount, 10)

await aliceClient.playMove(alice, gameIndex, { x: 1, y: 2 }, { x: 2, y: 3 }, "auto")

expect(parseInt((await aliceClient.getBalance(alice, "token")).amount, 10)).to.be.equal(

aliceBalBefore - 1,

)

const bobBalBefore = parseInt((await aliceClient.getBalance(bob, "token")).amount, 10)

await bobClient.playMove(bob, gameIndex, { x: 0, y: 5 }, { x: 1, y: 4 }, "auto")

expect(parseInt((await aliceClient.getBalance(bob, "token")).amount, 10)).to.be.equal(

bobBalBefore - 1,

)

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L114-L126" \t "_blank)

These first tests demonstrate the use of the createGame and playMove functions you created. These functions send a single message per transaction, and wait for the transaction to be included in a block before moving on.

In the next paragraphs you:

1. Will send many transactions in a single block.
2. Will send a transaction with more than one message in it.

The transaction with more than one message that you will send is where Alice, the black player, [captures two red pieces in two successive moves (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/player.ts#L39-L41). The checkers rules allow this.

The many transactions per block will be those that make the game reach that point.

Prepare the game moves of a complete game:



Copy

export interface GameMove {

player: Player

from: Pos

to: Pos

}

export const completeGame: GameMove[] = [

{ player: "b", from: { x: 1, y: 2 }, to: { x: 2, y: 3 } },

{ player: "r", from: { x: 0, y: 5 }, to: { x: 1, y: 4 } },

{ player: "b", from: { x: 2, y: 3 }, to: { x: 0, y: 5 } },

{ player: "r", from: { x: 4, y: 5 }, to: { x: 3, y: 4 } },

{ player: "b", from: { x: 3, y: 2 }, to: { x: 2, y: 3 } },

{ player: "r", from: { x: 3, y: 4 }, to: { x: 1, y: 2 } },

{ player: "b", from: { x: 0, y: 1 }, to: { x: 2, y: 3 } },

{ player: "r", from: { x: 2, y: 5 }, to: { x: 3, y: 4 } },

{ player: "b", from: { x: 2, y: 3 }, to: { x: 4, y: 5 } },

{ player: "r", from: { x: 5, y: 6 }, to: { x: 3, y: 4 } },

{ player: "b", from: { x: 5, y: 2 }, to: { x: 4, y: 3 } },

{ player: "r", from: { x: 3, y: 4 }, to: { x: 5, y: 2 } },

{ player: "b", from: { x: 6, y: 1 }, to: { x: 4, y: 3 } },

{ player: "r", from: { x: 6, y: 5 }, to: { x: 5, y: 4 } },

{ player: "b", from: { x: 4, y: 3 }, to: { x: 6, y: 5 } },

{ player: "r", from: { x: 7, y: 6 }, to: { x: 5, y: 4 } },

{ player: "b", from: { x: 7, y: 2 }, to: { x: 6, y: 3 } },

{ player: "r", from: { x: 5, y: 4 }, to: { x: 7, y: 2 } },

{ player: "b", from: { x: 4, y: 1 }, to: { x: 3, y: 2 } },

{ player: "r", from: { x: 3, y: 6 }, to: { x: 4, y: 5 } },

{ player: "b", from: { x: 5, y: 0 }, to: { x: 4, y: 1 } },

{ player: "r", from: { x: 2, y: 7 }, to: { x: 3, y: 6 } },

{ player: "b", from: { x: 0, y: 5 }, to: { x: 2, y: 7 } },

{ player: "r", from: { x: 4, y: 5 }, to: { x: 3, y: 4 } },

{ player: "b", from: { x: 2, y: 7 }, to: { x: 4, y: 5 } },

// player captures again

{ player: "b", from: { x: 4, y: 5 }, to: { x: 2, y: 3 } },

{ player: "r", from: { x: 6, y: 7 }, to: { x: 5, y: 6 } },

{ player: "b", from: { x: 2, y: 3 }, to: { x: 3, y: 4 } },

{ player: "r", from: { x: 0, y: 7 }, to: { x: 1, y: 6 } },

{ player: "b", from: { x: 3, y: 2 }, to: { x: 4, y: 3 } },

{ player: "r", from: { x: 7, y: 2 }, to: { x: 6, y: 1 } },

{ player: "b", from: { x: 7, y: 0 }, to: { x: 5, y: 2 } },

{ player: "r", from: { x: 1, y: 6 }, to: { x: 2, y: 5 } },

{ player: "b", from: { x: 3, y: 4 }, to: { x: 1, y: 6 } },

{ player: "r", from: { x: 4, y: 7 }, to: { x: 3, y: 6 } },

{ player: "b", from: { x: 4, y: 3 }, to: { x: 3, y: 4 } },

{ player: "r", from: { x: 5, y: 6 }, to: { x: 4, y: 5 } },

{ player: "b", from: { x: 3, y: 4 }, to: { x: 5, y: 6 } },

{ player: "r", from: { x: 3, y: 6 }, to: { x: 2, y: 5 } },

{ player: "b", from: { x: 1, y: 6 }, to: { x: 3, y: 4 } },

]

Expand



src /

types /

checkers /

player.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/player.ts" \l "L8-L56" \t "_blank)



Note how you already played the first two.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#multiple-transactions-in-a-block) Multiple transactions in a block

You are going to send [22 transactions (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/player.ts#L17-L38) in as quick a succession as possible. If you waited for each to be included in a block, it would take you in the order of 22\*5 == 110 seconds. That's very long for a test. It is better to find a way to include more transactions per block.

You will face several difficulties when you want to send multiple transactions in a single block. The **first difficulty** is as follows:

1. Each transaction signed by an account must mention the correct sequence of that account at the time of inclusion in the block. This is to make sure the transactions of a given account are added in the right order and to prevent transaction replay.
2. After each transaction, this sequence number increments, ready to be used for the next transaction of the account.
3. The signing client's [signAndbroadcast (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/signingstargateclient.ts#L280) function [fetches the sequence (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/signingstargateclient.ts#L318-L325) number from the blockchain.

In other words, the signing client can only know about the transactions that have been included in a block. It has no idea whether there are already pending transactions with a higher sequence that would result in the account's sequence being higher when your poorly crafted transaction is checked, therefore causing it to be rejected.

Fortunately, the [sign (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/signingstargateclient.ts#L310) function can take [any sequence number (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/signingstargateclient.ts#L315). You will therefore force the sequence number to the one you know to be right eventually:

1. It will start at the number as fetched from the blockchain.
2. Whenever you sign a new transaction you will increment this sequence number and keep track of it in your own variable.

Because JavaScript has low assurances when it comes to threading, you need to make sure that each sign command happens after the previous one, or your sequence incrementing may get messed up. For that, you should not use Promise.all on something like array.forEach(() => { await }), which fires all promises roughly at the same time. Instead you will use a while() { await } pattern.

There is a **second difficulty** when you want to send that many signed transactions. The client's broadcastTx function [waits for it (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/stargateclient.ts#L420-L424) to be included in a block, which would defeat the purpose of signing separately. Fortunately, if you look into its content, you can see that it calls [this.forceGetTmClient().broadcastTxSync (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/stargate/src/stargateclient.ts#L410). This CometBFT client function returns only [the hash (opens new window)↗](https://github.com/cosmos/cosmjs/blob/v0.28.11/packages/tendermint-rpc/src/tendermint34/tendermint34client.ts#L172), that is *before any inclusion in a block*.

On the other hand, you want the last transaction to be included in the block so that when you query for the stored game you get the expected values. Therefore you will:

1. Send the first 21 signed transactions with the *fast* this.forceGetTmClient().broadcastTxSync.
2. Send the last signed transaction with a *slow* client broadcastTx.

Here again, you need to make sure that you submit all transactions in sequential manner, otherwise a player may in effect try to play before their turn. At this point, you trust that CometBFT includes the transactions in the order in which they were submitted. If CometBFT does any shuffling between Alice and Bob, you may end up with a "play before their turn" error.

With luck, all transactions may end up in a single block, which would make the test 22 times faster than if you had waited for each transaction to get into its own block.



You would use the same techniques if you wanted to stress test your blockchain. This is why these paragraphs are more than just entertainment.

Add a way to track the sequences of Alice and Bob:



Copy

interface ShortAccountInfo {

accountNumber: number

sequence: number

}

const getShortAccountInfo = async (who: string): Promise<ShortAccountInfo> => {

const accountInfo: Account = (await aliceClient.getAccount(who))!

return {

accountNumber: accountInfo.accountNumber,

sequence: accountInfo.sequence,

}

}

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L128-L138" \t "_blank)

Add helpers to pick the right Alice or Bob values:



Copy

const whoseClient = (who: Player) => (who == "b" ? aliceClient : bobClient)

const whoseAddress = (who: Player) => (who == "b" ? alice : bob)

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L139-L140" \t "_blank)

Add a function to your client to give you access to CometBFT's broadcastTxSync:



Copy

public async tmBroadcastTxSync(tx: Uint8Array): Promise<BroadcastTxSyncResponse> {

return this.forceGetTmClient().broadcastTxSync({ tx })

}

src /

checkers\_stargateclient.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/checkers_stargateclient.ts" \l "L23-L25" \t "_blank)

Note that this function is on the *read-only* Stargate client. The signing Stargate client also holds a signer, but because here the signing is taking place outside Stargate, it is reasonable to add tmBroadcastTxSync in the Stargate that has the easiest constructor.

Create your it test with the necessary initializations:



Copy

it("can continue the game up to before the double capture", async function () {

this.timeout(20\_000)

const client: CheckersStargateClient = await CheckersStargateClient.connect(RPC\_URL)

const chainId: string = await client.getChainId()

const accountInfo = {

b: await getShortAccountInfo(alice),

r: await getShortAccountInfo(bob),

}

// TODO

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L142-L149" \t "_blank)

Now get all 22 signed transactions, from index 2 to index 23:



Copy

const txList: TxRaw[] = []

let txIndex: number = 2

while (txIndex < 24) {

const gameMove: GameMove = completeGame[txIndex]

txList.push(

await whoseClient(gameMove.player).sign(

whoseAddress(gameMove.player),

[

{

typeUrl: typeUrlMsgPlayMove,

value: {

creator: whoseAddress(gameMove.player),

gameIndex: gameIndex,

fromX: gameMove.from.x,

fromY: gameMove.from.y,

toX: gameMove.to.x,

toY: gameMove.to.y,

},

},

],

{

amount: [{ denom: "stake", amount: "0" }],

gas: "500000",

},

`playing move ${txIndex}`,

{

accountNumber: accountInfo[gameMove.player].accountNumber,

sequence: accountInfo[gameMove.player].sequence++,

chainId: chainId,

},

),

)

txIndex++

}

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L150-L183" \t "_blank)



Note how:

1. The moves [0 and 1 (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/player.ts#L15-L16) already took place in the previous it test.
2. The gas fee can no longer be "auto" and has to be set, here at a price of 0. You may have to adjust this depending on your running chain.
3. The sequence number of the signer is increased **after** it has been used: .sequence++.
4. There is an await on .sign, but this is very fast because the signing happens without contacting the blockchain.
5. The message has to be built at a low level, bypassing the convenient playMove method.

With all the transactions signed, you can *fire* broadcast the first 21 of them:



Copy

const hashes: BroadcastTxSyncResponse[] = []

txIndex = 0

while (txIndex < txList.length - 1) {

const txRaw: TxRaw = txList[txIndex]

hashes.push(await client.tmBroadcastTxSync(TxRaw.encode(txRaw).finish()))

txIndex++

}

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L185-L191" \t "_blank)

You now *normally* broadcast the last one:



Copy

const lastDelivery: DeliverTxResponse = await client.broadcastTx(

TxRaw.encode(txList[txList.length - 1]).finish(),

)

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L193-L195" \t "_blank)

If you are interested, you can log the blocks in which the transactions were included:



Copy

console.log(

txList.length,

"transactions included in blocks from",

(await client.getTx(toHex(hashes[0].hash)))!.height,

"to",

lastDelivery.height,

)

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L197-L203" \t "_blank)

Lastly, make sure that the game has the expected board:



Copy

const game: StoredGame = (await checkers.getStoredGame(gameIndex))!

expect(game.board).to.equal("\*b\*b\*\*\*b|\*\*b\*b\*\*\*|\*\*\*b\*\*\*r|\*\*\*\*\*\*\*\*|\*\*\*r\*\*\*\*|\*\*\*\*\*\*\*\*|\*\*\*r\*\*\*\*|r\*B\*r\*r\*")

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L205-L206" \t "_blank)

You have now brought the game up to the point just before the black player can capture two pieces in two moves.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#combine-messages) Combine messages

Alice, the black player can capture [two pieces in one turn (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/src/types/checkers/player.ts#L39-L41). The checkers rules allow that.

You are now ready to send that one transaction with the two messages with the use of signAndBroadcast. Add an it test with the right initializations:



Copy

it("can send a double capture move", async function () {

this.timeout(10\_000)

const firstCaptureMove: GameMove = completeGame[24]

const secondCaptureMove: GameMove = completeGame[25]

// TODO

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L209-L212" \t "_blank)

In it, make the call with the correctly crafted messages.



Copy

const response: DeliverTxResponse = await aliceClient.signAndBroadcast(

alice,

[

{

typeUrl: typeUrlMsgPlayMove,

value: {

creator: alice,

gameIndex: gameIndex,

fromX: firstCaptureMove.from.x,

fromY: firstCaptureMove.from.y,

toX: firstCaptureMove.to.x,

toY: firstCaptureMove.to.y,

},

},

{

typeUrl: typeUrlMsgPlayMove,

value: {

creator: alice,

gameIndex: gameIndex,

fromX: secondCaptureMove.from.x,

fromY: secondCaptureMove.from.y,

toX: secondCaptureMove.to.x,

toY: secondCaptureMove.to.y,

},

},

],

"auto",

)

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L213-L240" \t "_blank)

Next, collect the events and confirm they match your expectations:



Copy

const logs: Log[] = JSON.parse(response.rawLog!)

expect(logs).to.be.length(2)

expect(getCapturedPos(getMovePlayedEvent(logs[0])!)).to.deep.equal({

x: 3,

y: 6,

})

expect(getCapturedPos(getMovePlayedEvent(logs[1])!)).to.deep.equal({

x: 3,

y: 4,

})

test /

integration /

stored-game-action.ts

[View source→](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts" \l "L241-L250" \t "_blank)



Note how it checks the logs for the captured attributes. In effect, a captured piece has x and y as the average of the respective from and to positions' fields.

Sending a single transaction with two moves is cheaper and faster, from the point of view of the player, than sending two separate ones for the same effect.



It is not possible for Alice, who is the creator and black player, to send in a single transaction both a message for creation *and* a message to make the first move on it. This is because the index of the game is not known before the transaction has been included in a block, and with that the index computed.



Of course, she could try to do this. However, if her move failed because of a wrong game id, then the whole transaction would revert, and that would include the game creation being reverted.   
  
Worse, a malicious attacker could **front-run** Alice's transaction with another transaction, creating a game where Alice is also the black player and whose id ends up being the one Alice signed in her first move. In the end she would make the first move on a game she did not really intend to play. This game could even have a wager that is *all* of Alice's token holdings.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#further-tests) Further tests

You can add further tests, for instance to see what happens with token balances when you continue playing the game [up to its completion (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/signing-stargate/test/integration/stored-game-action.ts#L253-L321).

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#prepare-your-checkers-chain) Prepare your checkers chain

If you launch the tests just like you did in the [previous section](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/1-cosmjs-objects.html), you may be missing a faucet.

Adjust what you did.

* If you came here after going through the rest of the hands-on exercise, you know how to launch a running chain with Ignite, which has a faucet to start with.
* If you arrived here and are only focused on learning CosmJS, it is possible to abstract away niceties of both the running chain and a faucet in a minimal package. For this, you need Docker and to create an image:
  1. Get the Dockerfile:



Copy

$ curl -O https://raw.githubusercontent.com/cosmos/b9-checkers-academy-draft/main/Dockerfile-standalone

* 1. Build the Docker images:

**Checkers**

**Faucet**



Copy

$ docker build . \

-f Dockerfile-standalone \

-t checkersd\_i:standalone

Copy

$ docker build . \

-f Dockerfile-standalone \

--target cosmos-faucet \

-t cosmos-faucet\_i:0.28.11

If you have another preferred method, make sure to keep track of the required RPC\_URL and FAUCET\_URL accordingly.



If you are curious about how this Dockerfile-standalone was created, head to the [run in production](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html) section.

[#Copy link](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#launch-the-tests) Launch the tests

Launch your checkers chain and the faucet. You can choose your preferred method, as long as they can be accessed at the RPC\_URL and FAUCET\_URL you defined earlier. For the purposes of this exercise, you have the choice between three methods:

**Docker standalone**

**Local Ignite**

**Docker Ignite**



Copy

$ docker network create checkers-net

$ docker run --rm -it \

-p 26657:26657 \

--name checkers \

--network checkers-net \

--detach \

checkersd\_i:standalone start

$ sleep 10

$ docker run --rm -it \

-p 4500:4500 \

--name cosmos-faucet \

--network checkers-net \

--detach \

cosmos-faucet\_i:0.28.11 start http://checkers:26657

$ sleep 20

If your checkers-net network already exists, the first command fails with:



Copy

Error response from daemon: network with name checkers-net already exists

But that is okay.



* The chain needs about 10 seconds to start listening on port 26657. The faucet does not *retry*, so you have to be sure the chain is started.
* The faucet container itself takes about 20 seconds to be operational, as it creates 4 transactions in 4 blocks.
* They are started in --detached mode, but you are free to start them attached in different shells.

Copy

$ ignite chain serve

Copy

$ docker network create checkers-net

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

-p 4500:4500 \

-p 26657:26657 \

--network checkers-net \

--name checkers \

--network-alias cosmos-faucet \

checkers\_i \

ignite chain serve

If your checkers-net network already exists, the first command fails with:

Copy

Error response from daemon: network with name checkers-net already exists

But that is okay.

An alias, cosmos-faucet, is added so that it can be accessed too.

Now you can run the tests:

**Local**

**Docker**



Copy

$ npm test

Copy

$ docker run --rm -it \

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

--network checkers-net \

--env RPC\_URL="http://checkers:26657" \

--env FAUCET\_URL="http://cosmos-faucet:4500" \

node:18.7-slim \

npm test

Note how RPC\_URL and FAUCET\_URL override the default values found in .env, typically localhost.

The only combination of running chain / running tests that will not work with the above is if you run Ignite on your local computer and the tests in a container. For this edge case, you should put your host IP address in RPC\_URL and FAUCET\_URL, for instance --env RPC\_URL="http://YOUR-HOST-IP:26657".

If you started the chain in Docker, when you are done you can stop the containers with:



Copy

$ docker stop cosmos-faucet checkers

$ docker network rm checkers-net

synopsis

To summarize, this section has explored:

* How to create the elements necessary for you to begin sending transactions to your checkers blockchain, including encodable messages, a signing client, and action methods that permit interaction with the blockchain.
* How to test your signing client, including key preparation (with either a mnemonic or a private key) and client preparation, followed by functions such as creating a game, or playing a move.
* How to send multiple transactions in one block, simulating a stress test.
* How to send multiple messages in a single transaction.

previous

[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/1-cosmjs-objects.html)

**[Create Custom Objects](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/1-cosmjs-objects.html)**

up next

**[Get an External GUI](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/3-external-gui.html)**

[[](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/3-external-gui.html)](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/3-external-gui.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

[Encodable messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#encodable-messages)

[A signing Stargate for checkers](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#a-signing-stargate-for-checkers)

[The action methods](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#the-action-methods)

[Integration tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#integration-tests)

[Key preparation](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#key-preparation)

[Token preparation](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#token-preparation)

[Adding tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#adding-tests)

[Create and play](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#create-and-play)

[Multiple transactions in a block](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#multiple-transactions-in-a-block)

[Combine messages](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#combine-messages)

[Further tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#further-tests)

[Prepare your checkers chain](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#prepare-your-checkers-chain)

[Launch the tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#launch-the-tests)

#### **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)