TxPipe Docs

How to use TX3 in a web project

> ⚠ **Work in progress**
>
> This tutorial is a work in progress and will be updated with more examples and details soon. Stay tuned for the complete guide on using TX3 in web projects!

Before diving into the tutorial, ensure you have the following prerequisites set up in your development environment:

Prerequisites

- **Node.js 20 or higher**: You can download Node.js from nodejs.org or use a version manager like `nvm`.
- **Trix**: The TX3 compiler and toolchain. This is the core tool that processes your TX3 files and generates the necessary code for your application.
- **Dolos (optional)**: The Dolos CLI is used to run the trp server locally but you can also use it from Demeter Platform.
- **VSCode Extension**: The TX3 VSCode extension provides robust features such as syntax highlighting, LSP diagnostics, and interactive panels for transaction resolution and

visualization. enhance your development experience. You can find it in the VSCode Marketplace.

> **Tip**
>
> **Trix**, **Dolos** and **VSCode Extension** requirements can be installed following the instructions in the TX3 documentation.

Asteria Joystick Example

We will start creating a simple web project that uses TX3 to manage transactions. This example will guide you through the process of setting up a basic web application, integrating TX3, and using it to handle transactions.

Create a new project

For the demo purposes we will use react-router v7 and create a new project using the following command:

```
npx create-react-router@latest --template remix-run/react-router-templates/default
```

By default, this command will create a new project and ask you to use npm as the package manager. If you prefer another package manager, you can specify it using the `--package-manager` option.

At this point we have a basic React project set up. You can run it using:

```
npm run dev
```

For the tutorial, we will use file route conventions, so we need to install `@react-router/fs-routes` package:

```
npm install @react-router/fs-routes
```

Update the `app/routes.ts` file to use the file route conventions. Here's an example of how to do this:

```
import { type RouteConfig } from "@react-router/dev/routes";
import { flatRoutes } from "@react-router/fs-routes";

export default flatRoutes() satisfies RouteConfig;
```

Rename `app/routes/home.tsx` to `app/routes/_index.tsx` to make it the default route for our application.

Finally, update the content of this route:

```
// Types
import type { Route } from './+types/_index';

export function meta(_: Route.MetaArgs) {
  return [
    { title: 'Asteria Joystick' },
    { name: 'description', content: 'Asteria controls using TX3' },
  ];
}

export default function Home() {
  return (
    <main className="flex items-center flex-col justify-center pt-16 pb-4
gap-4">
      <h1 className="text-3xl font-bold">Welcome to Asteria Joystick</h1>
    </main>
  );
}
```

Install TX3 dependencies

Next, we need to install the TX3 vite plugin and include it in our project. This plugin will allow us to process TX3 files and integrate them into our React application. Run the following command to install the plugin:

```

npm install --save-dev vite-plugin-tx3
```

Configure Vite

Now, we need to configure Vite to use the TX3 plugin. Open the `vite.config.js` file in your project and add the TX3 plugin to the Vite configuration. Here's an example of how to do this:

```
import { reactRouter } from "@react-router/dev/vite";
import tailwindcss from "@tailwindcss/vite";
import { defineConfig } from "vite";
import tsconfigPaths from "vite-tsconfig-paths";
import tx3Plugin from 'vite-plugin-tx3';

export default defineConfig({
  plugins: [tailwindcss(), reactRouter(), tsconfigPaths(), tx3Plugin()],
});
```

This plugin will handle the TX3 file in your project and bindgen it based on the trix.toml configuration.

Start a TX3 project

Now that we have the TX3 plugin set up, we can start a TX3 project. For this, will run trix init on root project directory

```
trix init
```

On bindgen step, we will select "typescript". This command will create a `trix.toml` file in your project directory, which is the configuration file for TX3.

In the `trix.toml` file, we will include the trp local server configuration and change the bindgen output directory to generate the TypeScript bindings in the `node_modules` folder.

```
[[bindings]]
plugin = "typescript"
output_dir = "./node_modules/.tx3"

[profiles.devnet.trp]
url= "http://localhost:8164"
headers = {}
```

This will define the default trp client configuration using `http://localhost:8164` as the URL for the local TRP server. Anyway, it's possible to create a trp client by code directly, we will see this later.

Updata main.tx3

We will update the `main.tx3` file to include the basic create ship transaction.

```
party Admin;
party Player;

policy ShipShardPolicyId =
0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20;
policy FuelPolicyId =
0xfc8ad4f84181b85dc04f7b8c2984b129284c4e272ef45cd6440575fd4655454c;

asset Fuel =
0xfc8ad4f84181b85dc04f7b8c2984b129284c4e272ef45cd6440575fd4655454c."FUEL";
asset AdminToken =
0x5ffc30389bee5838f5f25d015642f8d291769168145a80a686556e8a."Final Admin";

// Datums
type ShipDatum {
    pos_x: Int,
    pos_y: Int,
```

```
    ship_token_name: Bytes,
    pilot_token_name: Bytes,
    last_move_latest_time: Int,
}

// Asteria
type AsteriaDatum {
    ship_counter: Int,
    shipyard_policy: Bytes, // Is possible to define a maxLength?
}

tx createShip(
    p_pos_x: Int, // Ship Position X
    p_pos_y: Int, // Ship Position Y
    initial_fuel: Int, // Initial Fuel
    tx_latest_posix_time: Int,
    ship_mint_lovelace_fee: Int, // Lovelace fee for minting ship
    ship_name: Bytes, // Name of the ship
    pilot_name: Bytes, // Name of the pilot
) {
    // References
    // SpaceTime
    reference SpaceTimeRef {
        ref:
0x41e5881cd3bdc3f08bcf341796347e9027e3bcd8d58608b4fcfca5c16cbf5921#0,
    }

    // Pellet
    reference PelletRef {
        ref:
0xba6fab625d70a81f5d1b699e7efde4b74922d06224bef1f6b84f3adf0a61f3f3#0,
    }

    // Asteria
    reference AsteriaRef {
        ref:
```

```
0x39871aab15b7c5ab1075ba431d7475f3977fe40fbb8d654b6bdf6f6726659277#0,
    }

    // Input blocks
    // Ensure that the Player has the required assets
    input source {
        from: Player,
        min_amount: fees + Ada(ship_mint_lovelace_fee),
    }

    input asteria {
        // Asteria Contract
        from:
"addr_test1wqdsuy97njefz53rkhd4v6a2kuqk0md5mrn996ygwekrdyq369wjg",
        min_amount: AdminToken(1),
        datum_is: AsteriaDatum,
    }

    // Optional mint/burn blocks
    // mint {
    //     amount:
AnyAsset(0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20,
"PILOT" + asteria.ship_counter + 1, 1)
    //         +
AnyAsset(0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20,
"SHIP" + asteria.ship_counter + 1, 1),
    //     redeemer: (),
    // }
    mint {
        amount:
AnyAsset(0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20,
pilot_name, 1)
            +
AnyAsset(0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20,
ship_name, 1),
        redeemer: ()
```

```
        redeemer: (),
    }

    mint {
        amount: Fuel(initial_fuel),
        redeemer: (),
    }

    // // Output blocks
    output {
        to: Player,
        amount: source - fees - Ada(ship_mint_lovelace_fee) +
AnyAsset(0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20,
pilot_name, 1),
    }

    // // Output - Pay to Contract
    output {
        // Asteria Contract
        to:
"addr_test1wqdsuy97njefz53rkhd4v6a2kuqk0md5mrn996ygwekrdyq369wjg",
        amount: asteria + Ada(ship_mint_lovelace_fee) + AdminToken(1),
        datum: AsteriaDatum {
            ship_counter: asteria.ship_counter + 1,
            shipyard_policy: asteria.shipyard_policy,
        },
    }

    output {
        // SpaceTime Contract
        to:
"addr_test1wru5jl7xf6rufkjwcmft6x5rnd40zznhcyyp0km3gwkr6gq6sxzm6",
        amount:
AnyAsset(0xf9497fc64e87c4da4ec6d2bd1a839b6af10a77c10817db7143ac3d20,
ship_name, 1) + Fuel(initial_fuel),
        datum: ShipDatum {
```

```
            pos_x: p_pos_x,
            pos_y: p_pos_y,
            ship_token_name: ship_name,
            pilot_token_name: pilot_name,
            last_move_latest_time: tx_latest_posix_time,
        },
    }
}
```

> **Note**
>
> This is a simplified version of the end main.tx3 file, which includes the basic create ship transaction.

Install tx3-sdk

To install the TX3 SDK, run the following command in your project directory:

```
npm install tx3-sdk
```

This will add the TX3 SDK as a dependency in your project to interact with TX3 protocols.

Test our app is working

Now, we can test our application to ensure everything is set up correctly.

First, we will update `app/routes/_index.tsx` to include our protocol and a button to call the transaction.

1. Import our tx3 protocol and useFetcher from react-router

```
import { useFetcher } from 'react-router';

// Protocol
import { protocol } from '@tx3/protocol';
```

2 Define an action, this will receive parameters in future and call the protocol to generate a cbor

```
export async function action(_: Route.ActionArgs) {
    const cbor = await protocol.createShipTx({
        initialFuel: 480,
        player: 'player-address',
        pPosX: 20,
        pPosY: 20,
        shipMintLovelaceFee: 1_000_000,
        txLatestPosixTime: Date.now(),
        // This will be removed in future and use datum information
        pilotName: new TextEncoder().encode('Pilot Name'),
        shipName: new TextEncoder().encode('Ship Name'),
    });
    console.log(cbor);
    return null;
}
```

3 Update the main component to include a fetcher form and a button to submit the transaction

```
<fetcher.Form method="POST" className="flex flex-col gap-4">
    <button
        type="submit"
        className="px-4 py-2 bg-blue-500 text-white rounded hover:bg-
blue-600 cursor-pointer"
    >
        Submit
    </button>
</fetcher.Form>
```

At this point, we can run our application using the following command:

```
npm run dev
```

If everything is set up correctly, you should see and output like this in the console:

```
Reading template from https://github.com/tx3-lang/web-sdk
Typescript bindgen successful
  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

which indicates that the tx3 file was bindgen successful and the application is running. Now, you can open your browser and navigate to `http://localhost:5173/` to see the application in action and press “Submit” button to call the transaction.

> **Work in progress**