How to get started interacting with Avail

Overview

Interaction with Avail Network is the most important thing that developers can start doing to get started in this great journey of modularism. There are multiple ways to start interacting and in this tutorial, we will check out all of them. We will be running a Turing testnet and will be ineracting with tools like Avail-js, Avail-Subxt and similar.

First thing to do is to run an Avail Node. To learn more on how to run a node, check out the documentatiohere .

Generating an AppId

What is an AppId?

As a general-purpose base layer, Avail is designed to support many modular chains at the same time, providing consensus and data availability to all of them simultaneously. Avail headers contain an index that allows a given modular chain (or "application" in Avail terminology) to determine and download only the sections of a block that have data for that particular application.

How to generate a new AppId

1. Access the Website
2. : Open your web browser and go to theGenerator web application (opens in a new tab)
3. .
4. Account Detection / Selection
5. :
6.
    - The website will automatically detect any accounts linked via browser extensions.
7.
    - Ensure you have the relevant extension installed and are logged in.
8.
    - Select the account you wish to use for the AppId creation.
9. Input Application Name
10. : In the provided field, enter the name of your application. Make sure the name is unique and identifies your app.
11. Send Transaction
12. : Submit the transaction after entering the application name. This will involve a confirmation step through your browser extension.
13. Receive Your Application Id:
14.
    - Upon successful transaction completion, your AppId will be displayed.
15.
    - Note down the Id for future reference.

For more information on AppId, check out the documentationhere .

Tools and Libraries to interact with Avail

There are multiple tools and libraries that can be used to interact with Avail. We will cover each one of them in depth here.

Avail-js

Avail-js is a JavaScript library that can be used to interact with Avail. It is a wrapper around the Substrate API and provides a simple interface to interact with Avail. To learn more about Avail-js from the github repo, checkout the repositoryhere (opens in a new tab) . We will get into some quick examples and tutorials on how to use Avail-js here.

Pre-requisites for Avail-js:Node.js(opens in a new tab)

Now, you can install the latest stable version of the avail-js library by running the following command:

npm

install

avail-js-sdk

Examples

There are a lot of examples already in the repository. We will be looking at how some of these work. To do so, we will be running a Turing testnet and will be interacting with the network using Avail-js. You can choose to run a local node as well if you wish.

Once you fully run the node, you also need to populateconfig.ts file with seed and endpoint details. You can choose to change other things as well, but we will be focusing on these two for now.

How to connect

For the starters, we will runconnect.ts . This is also located in theexamples(opens in a new tab) folder.

```
import { initialize } from

"avail-js-sdk"

// Global import import { isConnected , disconnect } from

"avail-js-sdk/chain"

// Modular import import config from

"../../config"

/* * Example to connect to a chain and get the ApiPromise*/ const

main

=

async () => { const

api

=

await

initialize ( config .endpoint) const [ chain ,

nodeName ,

nodeVersion ] =

await

Promise .all ([ api . rpc . system .chain () , api . rpc . system .name () , api . rpc . system .version () , ])

console .log ( Connected to chain { chain } using { nodeName } and node version { nodeVersion } - is connected: { isConnected () }  , ) await

disconnect () process .exit ( 0 ) } main ()
```

How to transfer funds

To transfer funds, you can use the following code in TypeScript. Please make sure that the funds that you are sending is specified in theconfig.ts file.

```
import { getDecimals , initialize , formatNumberToBalance , getKeyringFromSeed , isValidAddress } from

"avail-js-sdk" import { ISubmittableResult } from

"@polkadot/types/types/extrinsic" import { H256 } from

"@polkadot/types/interfaces/runtime"

import config from

"../../config"

const

main

=
```

```
async () => { try { if ( ! isValidAddress ( config .recipient)) throw

new

Error ( "Invalid Recipient" )

const

api

=

await

initialize ( config .endpoint) const

keyring

=

getKeyringFromSeed ( config .seed) const

options

= { app_id :

0 , nonce :

- 1 } const

decimals

=

getDecimals (api) const

amount

=

formatNumberToBalance ( config .amount , decimals)

const

oldBalance :

any

=

await

api . query . system .account ( config .recipient) console .log (Balance before the transfer call: { oldBalance[ "data" ][ "free" ] .toHuman () }
)

// Transaction call const

txResult

=

await

new

Promise < ISubmittableResult

    ((res) => { api . tx .balances .transferKeepAlive ( config .recipient , amount) .signAndSend (keyring , options ,
    (result :

ISubmittableResult ) => { console .log (Tx status: { result .status } ) if ( result .isFinalized ||

result .isError) { res (result) } }) }) console .log ( `Tx Hash: { txResult .txHash as
```

H256 } , Block Hash: { txResult . status .asFinalized as

H256 } `)

// Error handling const

error

=

txResult .dispatchError if (txResult .isError) { console .log (Transaction was not executed) } else

if (error !=

undefined) { if (error .isModule) { const

decoded

=

api . registry .findMetaError (error .asModule) const { docs ,

name ,

section } = decoded console .log ({ section } . { name } : { docs .join (" ") }) } else { console .log (error .toString ()) } process .exit (1) }

const

newBalance :

any

=

await

api . query . system .account (config .recipient) console .log (Balance after the transfer call: { newBalance["data"]["free"] .toHuman () })

process .exit (0) } catch (err) { console .error (err) process .exit (1) } } main () These two examples are just the starters on what can be achieved by using avail-js. There are a lot of otherexamples(opens in a new tab) in the repository that you can check out.

Avail-SubXt

SubXt is a library for interacting with Substrate based nodes in Rust and WebAssembly. We have built Avail-SubXt to interact with Avail nodes. To learn more about Avail-SubXt from the github repo, checkout the repositoryhere (opens in a new tab) . We will get into some quick examples and tutorials on how to use Avail-SubXt here.

Quickstart Optimium