Read & Write data on Avail DA

BEFORE YOU BEGIN

We recommend you go through these two pages in our docs before proceeding:

1. Get Testnet Tokens
2. : To help you get set with someAVAIL
3. tokens on the Turing testnet.
4. Create an AppID
5. : The guide below focusses on reading & submitting data on a particular AppID. It will help
6. to be familiar with the concept.

Setting up the dev environment

In this guide we will use some dedicated libraries to interact with Avail DA. To set up a dev environment foravail-js , please follow the stepsoutlined here .

Submitting data to Avail DA

avail-js avail-rust avail-go AvailApps explorer 1. Create a file namedsubmit-data.ts 2. Import the dependencies fromavail-js-sdk 3. and create amain 4. function:

import { SDK , WaitFor , Keyring , TransactionOptions } from

"avail-js-sdk"

const

main

=

async () => {

} main () 1. Initialize a new instance of the SDK inside themain 2. function:

// Initialize the SDK with a public Turing testnet endpoint // You can always switch it out with your own endpoint const

providerEndpoint

=

"wss://turing-rpc.avail.so/ws" ; const

sdk

=

await

SDK .New (providerEndpoint) 1. Initialize a new wallet using a12-word seed phrase 2. , and configure the params for the transaction:

const

Alice

=

"This is a random seed phrase please replace with your own" ; const

account

=

new

Keyring ({ type :

"sr25519" }) .addFromUri (Alice) const

data

=

"Example data to be submitted to Avail DA"

// You can submit data to any AppID of your choosing const

options :

TransactionOptions

= { app_id :

89 } 1. Submit the transaction by calling thedataAvailability_submitData 2. extrinsic via theavail-js 3. SDK:

//Submit the transaction const

result

=

await

sdk . tx . dataAvailability .submitData (data ,

WaitFor .BlockInclusion , account , options) if (result .isErr) { console .log (result .reason) process .exit (1) }

// Logging transaction details in the terminal console .log ("Data="

+

result . txData .data) console .log ("Who="

+

result . event .who +

", DataHash="

+

result . event .dataHash) console .log ("TxHash="

+

result .txHash +

", BlockHash="

+

result .blockHash) 1. Run the script using the following command:

ts-node

submit-data.ts 1. If everything went well, this is what your terminal should look like:

Reading data from Avail DA

You can read back your submitted data from Avail DA using theblockHash andtxHash of the transaction. 1. Create a file namedread-data.ts 2. Import the dependencies fromavail-js-sdk 3. and create amain 4. function:

import { initialize } from

"avail-js-sdk"

const

main

=

async () => { try {

} catch (err) { console .error (err) process .exit (1) }} main () 1. Initialize a new instance of the SDK inside themain 2. function, and declare the params:

//initialize sdk const

api

=

await

initialize ("wss://turing-rpc.avail.so/ws")

// Provide the transaction hash and block hash const [txHash ,

blockHash] = ["0x17463754ef4185f4faba2473535890e4397aa403830f3b5a77295340b9e7cf56" ,

"0x758036aa0db77bb34f6bf23b9fe290900f203ef4547e46c36fa486adbe6488e8"] console .log (Tx Hash: { txHash } , Block Hash: { blockHash }) 1. Extract the data:

// Extracting data const

block

=

await

api . rpc . chain .getBlock (blockHash) const

tx

=

block . block . extrinsics .find ((tx) =>

tx . hash .toHex () == txHash) if (tx ==

undefined) { console .log ("Failed to find the Submit Data transaction") process .exit (1) } 1. Parse the data to extract astring 2. :

console .log (JSON .stringify (tx)) const

dataHex

=

tx . method . args .map ((a) =>

a .toString ()) .join (", ") // Data retrieved from the extrinsic data let str =

"" for (let n =

0 ; n <

dataHex . length ; n +=

2) { str +=

String .fromCharCode (parseInt (dataHex .substring (n , n +

2) ,

16)) } console .log (This is the string that was submitted: { str }) 1. Run the script using the following command:

ts-node

read-data.ts 1. If everything went well, this is what your terminal should look like:

Complete example

The following code snippet combines the steps above into a single script to submit and fetch data from Avail:

```
import { initialize , getKeyringFromSeed } from
"avail-js-sdk" import { ISubmittableResult } from
"@polkadot/types/types/extrinsic" import { H256 } from
"@polkadot/types/interfaces/runtime" const
main
=
async () => { try { //initialize sdk const
api
=
await
initialize ( "wss://turing-rpc.avail.so/ws" ) // get your Avail account const
account
=
getKeyringFromSeed ( "This is a random seed phrase please replace with your own" ) console .log ( account .address)
// Data to be submitted const
data
=
"Hello World"
// submit the data using dataAvailability.submitData extrinsic const
txResult
=
await
new
Promise < ISubmittableResult
     ((res) => { api . tx . dataAvailability .submitData (data) .signAndSend (account , (result :
ISubmittableResult ) => { console .log (Tx status: { result .status } ) console .log (Block finalization and data retrieval can take upto 30
seconds...:                                                                   ) if ( result .isFinalized ||
result .isError) { res (result) } }) })
// Rejected Transaction handling if ( txResult .isError) { console .log (Transaction was not executed ) process .exit ( 1 ) }
// Passing the transaction hash and block hash to fetch the submitted data const [ txHash ,
blockHash ] = [ txResult .txHash as
H256 ,
txResult . status .asFinalized as
H256 ] console .log (Tx Hash: { txHash } , Block Hash: { blockHash } )
// Extracting data const
block

=

await

api . rpc . chain .getBlock (blockHash) const

tx

=

block . block . extrinsics .find ((tx) =>

tx . hash .toHex () ==

txHash .toHex ()) if (tx ==

undefined ) { console .log ( "Failed to find the Submit Data transaction" ) process .exit ( 1 ) }

console .log ( JSON .stringify (tx)) const

dataHex

=

tx . method . args .map ((a) =>

a .toString ()) .join ( ", " )

// Data retrieved from the extrinsic data let str =

"" for ( let n =

0 ; n <

dataHex . length ; n +=

2 ) { str +=

String .fromCharCode ( parseInt ( dataHex .substring (n , n +

2 ) ,

16 )) }

console .log ( submitted data: { str } ) process .exit () } catch (err) { console .error (err) process .exit ( 1 ) }} main ()
```

Estimate fees for your data submission

Theavail-js SDK provides a method to estimate the cost of submitting a particular piece of data to Avail DA. Here is how you can use it:

```
import { initialize , disconnect } from

"avail-js-sdk"

const

calculateCost

=

async () => { // Initialize the avail sdk providerEndpoint const

providerEndpoint

=

await

initialize ( "wss://turing-rpc.avail.so/ws" )
// Dummy sender - insert any address const
sender
=
"5CDGXH8Q9DzD3TnATTG6qm6f4yR1kbECBGUmh2XbEBQ8Jfa5"
//10^18 decimals to denominate to AVAIL const
DECIMAL
=
1000000000000000000
// Input the data let data =
"This is a random piece of string data!!!"
//Get the estimated cost in AVAIL const
cost
=
await
providerEndpoint . tx . dataAvailability .submitData (data) .paymentInfo (sender) const
costInAvail
= ( parseInt ( cost . partialFee .toString ()) / DECIMAL ) .toFixed ( 6 )
console .log ( Estimated Fees: { costInAvail } AVAIL )
await
disconnect () }
calculateCost () LOOKING FOR MORE EXAMPLES?
```

This page was written as an introduction to working with data submission/reading on Avail DA. In particular, we used thedataAvailability_submitData extrinsic to submit data to the network. Refer to ourAPI reference for more detailed examples.

Create an AppID Query Balances on Avail DA