Solana Dev 101 - Deserializing Account Data on Solana

Introduction

Interacting with data on Solana can be challenging. Account and transaction data are often encoded, which is good for efficiency but bad for developer sanity. Solana uses Borsh

(Binary Object Representation Serializer for Hashing) for its data serialization, which includes both the serialization and deserialization processes. One of the key advantages of Borsh is its determinism, ensuring consistent serialized output for the same input.

In this tutorial, we will go over how to deserialize account data from a token account, and return readable data you can utilize. You will use the simple example of breaking down raw account information from an NFT from its mint address using the Token Program library.

Below is a look in how we'll transform raw account data, to something more readable:

You can follow along to the full code in this tutorial by cloning our deserialize-account repository here.

Prerequisites

Here are the prerequisites for this tutorial:

- Basic TypeScript knowledge
- ts-node installed
- An RPC
- Example Repository

Environment Setup

1. Clone the example repository:
2. Navigate to the project directory:
3. Install npm:

Now you have the project setup! You can now jump into building out how you'll deserialize the account data for a given mint address.

Steps to Build

1. Fetching Account Data

1. In our /src/deserialize.ts file, let's import our needed modules:

You will use this later on to define our connection to Solana, and the PublicKey for the mint you want to deserialize the data for.

1. Below this, you'll set up the main function. You will also set up our Solana connection using our Helius RPC URL, and you can set the mint that you will be deserializing in the tutorial.

Make sure to replace api-key with your own Helius API key in the above. You can alternatively replace the mint used with an example of your own for this tutorial.

1. Next, you will set up a try/catch to fetch the raw account data for this mint:

In this step above, you are using our connection to make an RPC call on Solana for getAccountInfo

. This will return the initial data that you need to break down further. If no data is found, it will return null. Otherwise, it will console to your terminal with the results of the search.

You can now run ts-node deserialize to see the results. You should see a result similar to the following:

The purpose of being able to deserialize here would be to make this in a readable format. To do that as a developer, you would have to go to the source code here to find the layout expected to be given by the program it is created by.

2. Account Types Setup

In this example, you want to get the AccountInfo for an SPL mint 6MWfAt3S9Xu4ybxxgPm6e4LSwuXfyAwGXd5yfUqpox9K. To do this, you will have to break down the types for the raw mint data and buffer layout given to us in the Solana Program Library. Understanding the structure of the given data is a crucial step in deserializing any data on Solana.

This above is giving you the programs defined RawMint struct and MintLayout. You can actually just copy these over to our types file to use. Now, head over to our ./src/types directory.

1. Now, in the /src/types.ts file, and set up the import for our types:

This will define the above raw mint format and layout needed to deserialize the account data for the NFT in this example.

1. You can now set up our interface and layour similar to the above, note that you are using our imported PublicKey, u32, u8, publicKey, u64, and bool:

You have now set up our types for the raw account information! You can now import this in our main deserialize.ts file and use it to deserialize the data you returned earlier.

Note that in the MintLayout that the RawMint is defined as the struct. This is essentially typing the interface you set up to this layout defined in the GitHub source.

3. Deserialize the Returned Data

Now that you understand the structure of the expected data, you can set up our decode function, which is just a single line of code. You can return to the src/deserialize.ts file to set this up.

1. First, in our main deserialize.ts file, import our MintLayout from our types.ts file:
2. Now, you can just add a single line to our deserialize function, right below where you are fetching the account data:

This is using our MintLayout

to decode the data returned in our deserializeMint

function.

You can run ts-node deserialize from your ./src folder, and will get a result similar to the following:

This is definitely more readable! You can make this a bit more neat by breaking down the response in the next step to the types in the response.

1. Finally, you can still clean this data up here by adjusting the response to what is shown above. You can accomplish this by doing the following:

In the above code, you are only needing the convert the PublicKeys returned in a toString format. Otherwise the data can be returned in the format it is received in. Since the data format is expected, we can set this up ahead of time as well.

Which will return the following:

You can adjust this data in whichever way you'd like. This is simply breaking it down into a form where you can read the results better.

Full code:

Take a look at the full code for deserialize.ts here:

Conclusion

You have now deserialized NFT account data on Solana! You can apply the same methods to other use cases, and use similar research methods to figure out the structure of the program for the given data.

Make sure to check the source of the program from which you want to deserialize data, and try matching these methods to accomplish this on your own.

Resources

Borsh

Token Program Library

Solana Cookbook