JS/TS Client

Anchor provides a Typescript client library (@coral-xyz/anchor) that simplifies the process of interacting with Solana programs from the client in JavaScript or TypeScript.

Client Program#

To use the client library, first create an instance of aProgram using theIDL file generated by Anchor.

Creating an instance of theProgram requires the program's IDL and anAnchorProvider . AnAnchorProvider is an abstraction that combines two things:

- Connection
-
    - the connection to aSolana cluster
- (i.e. localhost, devnet, mainnet)
- Wallet
-
    - (optional) a default wallet used to pay and sign transactions

Frontend/Node Test File When integrating with a frontend using thewallet adapter , you'll need to set up theAnchorProvider andProgram .

import { Program, AnchorProvider, setProvider } from "@coral-xyz/anchor" ; import { useAnchorWallet, useConnection } from "@solana/wallet-adapter-react" ; import type { HelloAnchor } from "./idlType" ; import idl from "./idl.json" ;

const { connection } = useConnection (); const wallet = useAnchorWallet ();

const provider = new AnchorProvider (connection, wallet, {}); setProvider (provider);

export const program = new Program (idl as HelloAnchor , { connection, }); In the code snippet above:

- idl.json
- is the IDL file generated by Anchor, found at/target/idl/.json
- in an Anchor project.
- idlType.ts
- is the IDL type (for use with TS), found at/target/types/.ts
- in an Anchor project.

Alternatively, you can create an instance of theProgram using only the IDL and theConnection to a Solana cluster. This means there is no defaultWallet , but allows you to use theProgram to fetch accounts or build instructions without a connected wallet.

import { clusterApiUrl, Connection, PublicKey } from "@solana/web3.js" ; import { Program } from "@coral-xyz/anchor" ; import type { HelloAnchor } from "./idlType" ; import idl from "./idl.json" ;

const connection = new Connection (clusterApiUrl ("devnet"), "confirmed");

export const program = new Program (idl as HelloAnchor , { connection, });

Invoke Instructions#

Once theProgram is set up using a program IDL, you can use the AnchorMethodsBuilder to:

- Build individual instructions
- Build transactions
- Build and send transactions

The basic format looks like the following:

methods instruction accounts signers program.methods - This is the builder API for creating instruction calls from the program's IDL

await program. methods . instructionName (instructionData) . accounts ({}) . signers ([]) . rpc (); Anchor provides multiple methods for building program instructions:

.rpc .transaction .instruction Therpc() methodsends a signed transaction with the specified instruction and returns aTransactionSignature .

When using.rpc , theWallet from theProvider is automatically included as a signer.

// Generate keypair for the new account const newAccountKp = new Keypair ();

const data = new BN (42); const transactionSignature = await program.methods . initialize (data) . accounts ({ newAccount: newAccountKp.publicKey, signer: wallet.publicKey, systemProgram: SystemProgram.programId, }) . signers ([newAccountKp]) . rpc ();

Fetch Accounts#

TheProgram client simplifies the process of fetching and deserializing accounts created by your Anchor program.

Useprogram.account followed by the name of the account type defined in the IDL. Anchor provides multiple methods for fetching accounts.

all memcmp fetch fetchMultiple Useall() to fetch all existing accounts for a specific account type.

const accounts = await program.account.newAccount. all ();

Previous «IDL File Next PDAs with Anchor»