IDL File

An Interface Description Language (IDL) file provides a standardized JSON file describing the program's instructions and accounts. This file simplifies the process of integrating your on-chain program with client applications.

Key Benefits of the IDL:

- Standardization: Provides a consistent format for describing the program's
- instructions and accounts
- Client Generation: Used to generate client code to interact with the program

Theanchor build command generates an IDL file located at/target/idl/json .

The code snippets below highlights how the program, IDL, and client relate to each other.

Program Instructions#

Theinstructions array in the IDL corresponds directly to the instructions defined in your program. It specifies the required accounts and parameters for each instruction.

Program IDL Client The program below includes aninitialize instruction, specifying the accounts and parameters it requires.

use anchor_lang :: prelude ::* ;

declare_id! ("BYFW1vhC1ohxwRbYoLbAWs86STa25i9sD5uEusVjTYNd");

[program]

mod hello_anchor { use super ::* ; pub fn initialize (ctx : Context < Initialize

> , data : u64) -> Result <()> { ctx . accounts . new_account . data = data; msg! ("Changed data to: {}!" , data); Ok (()) } }

[derive(

Accounts)] pub struct Initialize <' info

> {

[account(init, payer

= signer, space = 8 + 8)] pub new_account : Account <' info , NewAccount

> ,

[account(

mut)] pub signer : Signer <' info

> , pub system_program : Program <' info , System , }

[account]

pub struct NewAccount { data : u64 , }

Program Accounts#

Theaccounts array in the IDL corresponds to the structs in a program annotated with the#[account] macro. These structs define the data stored in accounts created by the program.

Program IDL Client The program below defines aNewAccount struct with a singledata field of typeu64 .

use anchor_lang :: prelude ::* ;

declare_id! ("BYFW1vhC1ohxwRbYoLbAWs86STa25i9sD5uEusVjTYNd");

[program]

mod hello_anchor { use super ::* ; pub fn initialize (ctx : Context < Initialize

    , data : u64) -> Result <()> { ctx . accounts . new_account . data = data; msg! ("Changed data to: {}!" , data); Ok (()) } }

[derive(

Accounts)] pub struct Initialize <' info

    {

[account(init, payer

= signer, space = 8 + 8)] pub new_account : Account <' info , NewAccount

    ,

[account(

mut)] pub signer : Signer <' info

    , pub system_program : Program <' info , System , }

[account]

pub struct NewAccount { data : u64 , }

Discriminators#

Anchor assigns a unique 8 byte discriminator to each instruction and account type in a program. These discriminators serve as identifiers to distinguish between different instructions or account types.

The discriminator is generated using the first 8 bytes of the Sha256 hash of a prefix combined with the instruction or account name. As of Anchor v0.30, these discriminators are included in the IDL file.

Note that when working with Anchor, you typically won't need to interact directly with these discriminators. This section is primarily to provide context on how the discriminator is generated and used.

Instructions Accounts The instruction discriminator is used by the program to determine which specific instruction to execute when called.

When an Anchor program instruction is invoked, the discriminator is included as the first 8 bytes of the instruction data. This is done automatically by the Anchor client.

IDL "instructions" : [{ "name" : "initialize" , "discriminator" : [175 , 175 , 109 , 31 , 13 , 152 , 155 , 237], ... }] The discriminator for an instruction is the first 8 bytes of the Sha256 hash of the prefixglobal plus the instruction name.

For example:

sha256("global:initialize") Hexadecimal output:

af af 6d 1f 0d 98 9b ed d4 6a 95 07 32 81 ad c2 1b b5 e0 e1 d7 73 b2 fb bd 7a b5 04 cd d4 aa 30 The first 8 bytes are used as the discriminator for the instruction.

af = 175 af = 175 6d = 109 1f = 31 0d = 13 98 = 152 9b = 155 ed = 237 You can find the implementation of the discriminator generation in the Anchor codebasehere , which is usedhere .

Previous «Program Structure Next JS/TS Client»