# PDAs with Anchor

Program Derived Addresses (PDA) refer to a feature of Solana development that allows you to create a unique address derived deterministically from pre-defined inputs (seeds) and a program ID.

This section will cover basic examples of how to use PDAs in an Anchor program.

## Anchor PDA Constraints#

When using PDAs in an Anchor program, you generally use Anchor's account constraints to define the seeds used to derive the PDA. These constraints serve as security checks to ensure that the correct address is derived.

The constraints used to define the PDA seeds include:

- seeds
- : An array of optional seeds used to derive the PDA. Seeds can be
- static values or dynamic references to account data.
- bump
- : The bump seed used to derive the PDA. Used to ensure the address falls
- off the Ed25519 curve and is a valid PDA.
- seeds::program
-
    - (Optional) The program ID used to derive the PDA address.
- This constraint is only used to derive a PDA where the program ID is not the
- current program.

Theseeds andbump constraints are required to be used together.

### Usage Examples#

Below are examples demonstrating how to use PDA constraints in an Anchor program.

seeds bump seeds::program init Theseeds constraint specifies the optional values used to derive the PDA.

#### No Optional Seeds#

- Use an empty array[]
- to define a PDA without optional seeds.

# [derive(

Accounts )] pub struct InstructionAccounts <' info

{

# [account(

# seeds

[], bump, )] pub pda_account : SystemAccount <' info

, }

#### Single Static Seed#

- Specify optional seeds in theseeds
- constraint.

# [derive(

Accounts )] pub struct InstructionAccounts <' info

{

# [account(

# seeds

[ b"hello_world" ], bump, )] pub pda_account : SystemAccount <' info

, }

**Multiple Seeds and Account References#**

- Multiple seeds can be specified in theseeds
- constraint. Theseeds
- constraint can also reference other account addresses or account data.

# [derive(

Accounts )] pub struct InstructionAccounts <' info

{ pub signer : Signer <' info ,

# [account(

# seeds

[ b"hello_world" , signer . key() . as_ref()], bump, )] pub pda_account : SystemAccount <' info

, } The example above uses both a static seed (b"hello_world" ) and a dynamic seed (the signer's public key).

## PDA seeds in the IDL#

Program Derived Address (PDA) seeds defined in theseeds constraint are included in the program's IDL file. This allows the Anchor client to automatically resolve accounts using these seeds when constructing instructions.

This example below shows the relationship between the program, IDL, and client.

Program IDL Client The program below defines apda_account using a static seed (b"hello_world" ) and the signer's public key as a dynamic seed.

use anchor_lang :: prelude ::* ;

declare_id! ( "BZLiJ62bzRryYp9mRobz47uA66WDgtfTXhhgM25tJyx5" );

# [program]

mod hello_anchor { use super ::* ; pub fn test_instruction (ctx : Context < InstructionAccounts

) -> Result <()> { msg! ( "PDA: {}" , ctx . accounts . pda_account . key ()); Ok (()) } }

# [derive(

Accounts )] pub struct InstructionAccounts <' info

{ pub

signer : Signer <' info

,

# [account(

# seeds

[ b"hello_world" , signer . key() . as_ref()], bump, )] pub pda_account : SystemAccount <' info

, }

Previous «JS/TS Client Next CPIs with Anchor»