Using Data Feeds Onchain (Solana)

Chainlink Data Feeds are the quickest way to connect your smart contracts to the real-world market prices of assets. This guide demonstrates how to deploy a program to the Solana Devnet cluster and access Data Feeds onchain using theChainlink Solana Starter Kit . To learn how to read price feed data using offchain applications, see theUsing Data Feeds Offchain guide.

To get the full list of available Chainlink Data Feeds on Solana, see theSolana Feeds page. View the program that owns the Chainlink Data Feeds in theSolana Devnet Explorer , or theSolana Mainnet Explorer .

Select quality data feeds

Be aware of the quality of the data that you useLearn more about making responsible data quality decisions.

The Chainlink Data Feeds OCR2 program

The program that owns the data feeds on both Devnet and Mainnet isHEvSKofvBgfaexv23kMabbYqxasxU3mQ4ibBMEmJWHny . This is the program ID that you use to retrieve Chainlink Price Data onchain in your program. The source code for this program is available in thesmartcontractkit/chainlink-solana repository on GitHub.

You canadd data feeds to an existing project oruse the Solana Starter Kit .

Adding Data Feeds onchain in an existing project

You can read Chainlink Data Feed data onchain in your existing project using theChainlink Solana Crate .

Reading feed data

Although you can directly query the data feed accounts, you should not rely on the memory layout always being the same as it currently is. Based on this, the recommendation is to always use the consumer library queries below.

Import the Chainlink Solana Crate into your project and use the code sample to make function calls.

1. Add the Chainlink Solana Crate as an entry in yourCargo.tomlfile dependencies section, as shown in thestarter kit Cargo.toml example.

[dependencies]chainlink_solana="1.0.0" 2. Use the following code sample to query price data. Each function call to the Chainlink Solana library takes two parameters:

- Thefeed account that you want to query.
- TheChainlink Data Feeds OCR2 Program for the network. This is a static value that never changes.

The code sample has the following components:

- latest_round_data: Returns the latest round information for the specified price pair including the latest price
- description: Returns a price pair description such as SOL/USD
- decimals: Returns the precision of the price, as in how many numbers the price is padded out to
- Display: A helper function that formats the padded out price data into a human-readable price

/ *** THIS IS EXAMPLE CODE THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS EXAMPLE CODE THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION. */use chainlink_solanaaschainlink;use solana_program::{account_info::{next_account_info,AccountInfo},entrypoint,entrypoint::ProgramResult,msg,pubkey::Pubkey,};structDecimal{pub value:i128,pub decimals:u32,}impl Decimal{pub fnnew(value:i128,decimals:u32)->Self{Decimal{value,decimals}}}impl std::fmt::DisplayforDecimal{fnfmt(&self,f:&mut std::fmt::Formatter<'_>)->std::fmt::Result{letmut scaled_val=self.value.to_string();ifscaled_val.len()<=self.decimalsasusize{scaled_val.insert_str(0,&vec!["0";self.decimalsasusize-scaled_val.len()].join(""),);scaled_val.insert_str(0,"0.");}else{scaled_val.insert(scaled_val.len()-self.decimalsasusize,'.');}f.write_str(&scaled_val)}}// Declare and export the program's entrypointentrypoint!(process_instruction);// Program entrypoint's implementationpub fnprocess_instruction(_program_id:&Pubkey,// Ignoredaccounts:&[AccountInfo],_instruction_data:&[u8],// Ignored)->ProgramResult{msg!("Chainlink Price Feed Consumer entrypoint");letaccounts_iter=&mut accounts.iter();// This is the account of the price feed data to read fromletfeed_account=next_account_info(accounts_iter)?;// This is the chainlink solana program IDletchainlink_program=next_account_info(accounts_iter)?;letround=chainlink::latest_round_data(chainlink_program.clone(),feed_account.clone(),)?;letdescription=chainlink::description(("{} price is {}",description,decimal_print);Ok(())}**/ * THIS IS EXAMPLE CODE THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS EXAMPLE CODE THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN PRODUCTION. */use anchor_lang::prelude::*;use chainlink_solanaaschainlink;//Program ID required by Anchor. Replace with your unique program ID once you build your projectdeclare_id!("HPuUpM1bKbaqx7yY2EJ4hGBaA3QsfP5cofHHK99daz85");#[account]pubstructDecimal{pub value:i128,pub decimals:u32,}impl Decimal{pub fnnew(value:i128,decimals:u32)->Self{Decimal{value,decimals}}}impl std::fmt::DisplayforDecimal{fnfmt(&self,f:&mut std::fmt::Formatter<'_>)->std::fmt::Result{letmut scaled_val=self.value.to_string();ifscaled_val.len()<=self.decimalsasusize{scaled_val.insert_str(0,&vec!["0";self.decimalsasusize-scaled_val.len()].join(""),);scaled_val.insert_str(0,"0.");}else{scaled_val.insert(scaled_val.len()-self.decimalsasusize,'.');}f.write_str(&scaled_val)}}#[program]pub mod chainlink_solana_demo{use super::*;pub fnexecute(ctx:Context)->Result<()>{letround=chainlink::latest_round_data(ctx.accounts.chainlink_program.to_account_info(),ctx.accounts.chainlink_feed.to_account_info(),)?;letdescription=chainlink::description(ctx.accounts.chainlink_prog write the latest price to the program outputletdecimal_print=Decimal::new(round.answer,u32::from(decimals));msg!("{} price is {}",description,decimal_print);Ok(())}}#[derive(Accounts)]pubstructExecute<'info>{/// CHECK: We're reading data from this chainlink feed accountpub chainlink_feed:AccountInfo<'info>,/// CHECK: This is the Chainlink program librarypub chainlink_program:AccountInfo<'info>}Program Transaction logs:

RustRust with Anchor>Program logged:"Chainlink Price Feed Consumer entrypoint">Program logged:"SOL / USD price is 83.99000000">Program consumed:95953of1400000computeunits>Program return: HNYSbr77Jc9LhHeb9tx53SrWbWfNBnQzQrM4b3BB3PCRCA==Fetching transaction logs...['Program HEvSKofvBgfaexv23kMabbYqxasxU3mQ4ibBMEmJWHny consumed 1826 of 1306895 compute units','Program return: HEvSKofvBgfaexv23kMabbYqxasxU3mQ4ibBMEmJWHny CA==','Program HEvSKofvBgfaexv23kMabbYqxasxU3mQ4ibBMEmJWHny success','Program log: SOL / USD price is 93.76988029',] To learn more about Solana and Anchor, see theSolana Documentation and theAnchor Documentation .

Using the Solana starter kit

This guide demonstrates the following tasks:

- Write and deploy programs to theSolana Devnet cluster using Anchor.
- Retrieve price data data using theSolana Web3 JavaScript API with Node.js.

This example shows a full end to end example of using Chainlink Price Feeds on Solana. It includes an onchain program written in rust, as well as an offchain client written in JavaScript. The client passes in an account to the program, the program then looks up the latest price of the specified price feed account, and then stores the result in the passed in account. The offchain client then reads the value stored in the account.

Install the required tools

Before you begin, set up your environment for development on Solana:

1. InstallGit if it is not already configured on your system.
2. InstallNode.js 14 or higher . Runnode --versionto verify which version you have installed:

node--version 3. InstallYarn to simplify package management and run code samples. 4. Install a C compiler such as the one included inGCC . Some of the dependencies require a C compiler. 5. InstallRust :

curl--proto'=https'--tlsv1.2-sSfhttps://sh.rustup.rs|sh&&source$HOME/.cargo/env 6. Install the latest Mainnet version ofthe Solana CLI and export the path to the CLI:

sh-c"$(curl-sSfLhttps://release.solana.com/v1.13.6/install)"&&exportPATH="~/.local/share/solana/install/active_release/bin:$PATH"Runsolana --versionto make sure the Solana CLI is installed correctly.

solana--version 7. Install Anchor . On some operating systems, you might need to build and install Anchor locally. See theAnchor documentation for instructions.

After you install the required tools, build and deploy the example program from thesolana-starter-kit repository.

Deploy the example program

This example includes a contract written in Rust. Deploy the contract to the Solana Devnet cluster.

1. In a terminal, clone thesolana-starter-kit repository and change to thesolana-starter-kitdirectory:

gitclone https://github.com/smartcontractkit/solana-starter-kit&&cd./solana-starter-kitYou can see the complete code for the example onGitHub . 2. In the./solana-starter-kitdirectory, install Node.js dependencies defined in thepackage.jsonfile:

yarninstall 3. Create a temporary Solana wallet to use for this example. Use a temporary wallet to isolate development from your other wallets and prevent you from unintentionally using lamports on the Solana Mainnet. Alternatively, if you have an existing wallet that you want to use, locate the path to yourkeypair file and use it as the keypair for the rest of this guide.

solana-keygen new--outfile./id.jsonWhen you build your production applications and deploy Solana programs to the Mainnet cluster, always follow the security best practices in theSolana Wallet Guide for managing your wallets and keypairs. 4. Fund your Solana wallet. On Devnet, usesolana airdropto add tokens to your account. The contract requires at least 4 SOL to deploy and the faucet limits each request to 2 SOL, so you must make two requests to get a total of 4 SOL on your wallet:

solana airdrop2--keypair./id.json--urldevnet&&solana airdrop2--keypair./id.json--urldevnet* If the command line faucet does not work, runsolana addresson the temporary wallet to print the public key value for the wallet and request tokens fromSolFaucet :

solana address-k./id.json 5. Runanchor buildto build the example program. If you receive theno such subcommand: 'build-bpf'error, restart your terminal session and runanchor buildagain:

anchor build 6. The build process generates the keypair for your program's account. Before you deploy your program, you must add this public key to thelib.rsfile:

1. Get the keypair from the./target/deploy/chainlink_solana_demo-keypair.jsonfile that Anchor generated:

solana address-k./target/deploy/chainlink_solana_demo-keypair.json 2. Edit the./programs/chainlink_solana_demo/src/lib.rsfile and replace the keypair in thedeclare_id!()definition:

vi./programs/chainlink_solana_demo/src/lib.rsdeclare_id!("JC16qi56dgcLoaTVe4BvnCoDL6FhH5NtahA7jmWZFdqm"); 7. With the new program ID added, runanchor buildagain. This recreates the necessary program files with the correct program ID:

anchor build 8. Runanchor deployto deploy the program to the Solana Devnet. Remember to specify the keypair file for your wallet and override the default. This wallet is theaccount owner (authority) for the program:

anchor deploy--provider.wallet./id.json--provider.clusterdevnet 9. To confirm that the program deployed correctly, runsolana program show --programsto get a list of deployed programs that your wallet owns. For this example, check the list of deployed programs for theid.jsonwallet on the Solana Devnet:

solana program show--programs--keypair./id.json--urldevnetThe command prints the program ID, slot number, the wallet address that owns the program, and the program balance:

Program Id|Slot|Authority|Balance GRt21UnJFHZvcaWLbcUrXaTCFMREewDrm1DweDYBak3Z|110801571|FsQPnANKDhqpoayxCL3oDHFCBmrhP34NrfbDR34qbQUt|3.07874904SOLTo see additional details of your deployed program, copy the program ID and look it up in theSolana Devnet Explorer .

Now that the program is onchain, you can call it.

Call the deployed program

Use your deployed program to retrieve price data from a Chainlink data feed on Solana Devnet. For this example, call your deployed program using theclient.jsexample code.

1. Set theAnchor environment variables . Anchor uses these to determine which wallet to use and Solana cluster to use.

exportANCHOR_PROVIDER_URL=https://api.devnet.solana.com&&exportANCHOR_WALLET=./id.json 2. Run theclient.jsexample and pass the program address in using the--programflag:

nodeclient.js--program$(solana address-k./target/deploy/chainlink_solana_demo-keypair.json)If the script executes correctly, you will see output with the current price of SOL / USD.

⋮ Price Is:96.79778375Success ⋮ 3. Each request costs an amount of SOL that is subtracted from theid.jsonwallet. Runsolana balanceto check the remaining balance for your temporary wallet on Devnet.

solana balance--keypair./id.json--urldevnet 4. To get prices for a different asset pair, runclient.jsagain and add the--feedflag with one of the availableChainlink data feeds . For example, to get the price of BTC / USD on Devnet, use the following command:

nodeclient.js--program$(solana address-k./target/deploy/chainlink_solana_demo-keypair.json)--feedCzZQBrJCLqjXRfMjRN3fhbxur2QYHUzkpaRwkWsiPqbzPrice Is:12.4215826Success

The program that owns the data feeds isHEvSKofvBgfaexv23kMabbYqxasxU3mQ4ibBMEmJWHny , which you can see defined forconst CHAINLINK_PROGRAM_IDin theclient.jsfile.

Clean up

After you are done with your deployed contract and no longer need it, it is nice to close the program and withdraw the Devnet SOL tokens for future use. In a production environment, you will want to withdraw unused SOL tokens from any Solana program that you no longer plan to use, so it is good to practice the process when you are done with programs on Devnet.

1. Runsolana program showto see the list of deployed programs that your wallet owns and the balances for each of those programs:

solana program show--programs--keypair./id.json--urldevnetProgram Id|Slot|Authority|Balance GRt21UnJFHZvcaWLbcUrXaTCFMREewDrm1DweDYBak3Z|110801571|FsQPnANKDhqpoayxCL3oDHFCBmrhP34NrfbDR34qbQUt|3.07874904SOL 2. Runsolana program closeand specify the program that you want to close:

solana program close[YOUR_PROGRAM_ID]--keypair./id.json--urldevnetThe program closes and the remaining SOL is transferred to your temporary wallet. 3. If you have deployments that failed, they might still be in the buffer holding SOL tokens. Runsolana program showagain with the--buffersflag:

solana program show--buffers--keypair./id.json--urldevnetIf you have open buffers, they will appear in the list.

Buffer Address|Authority|Balance CSc9hnBqYJoYtBgsryJAmrjAE6vZ918qaFhL6N6BdEmB|FsQPnANKDhqpoayxCL3oDHFCBmrhP34NrfbDR34qbQUt|1.28936088SOL 4. If you have any buffers that you do not plan to finish deploying, run the samesolana program closecommand to close them and retrieve the unused SOL tokens:

solana program close[YOUR_PROGRAM_ID]--keypair./id.json--urldevnet 5. Check the balance on your temporary wallet.

solana balance--keypair./id.json--urldevnet 6. If you are done using this wallet for examples and testing, you can usesolana transfer to send the remaining SOL tokens to your default wallet or another Solana wallet that you use. For example, if your default wallet keypair is at~/.config/solana/id.json, you can sendALLof the temporary wallet's balance with the following command:

solana transfer ~/.config/solana/id.json ALL--keypair./id.json--urldevnetAlternatively, you can send the remaining balance to a web wallet. Specify the public key for your wallet instead of the path the default wallet keypair. Now you can use those Devnet funds for other examples and development.

To learn more about Solana and Anchor, see theSolana Documentation and theAnchor Documentation .