Accounts and Storing State

Storing State between Transactions#

If the program needs to store state between transactions, it does so usingaccounts . Accounts are similar to files in operating systems such as Linux in that they may hold arbitrary data that persists beyond the lifetime of a program. Also like a file, an account includes metadata that tells the runtime who is allowed to access the data and how.

Unlike a file, the account includes metadata for the lifetime of the file. That lifetime is expressed by a number of fractional native tokens calledlamports . Accounts are held in validator memory and pay"rent" to stay there. Each validator periodically scans all accounts and collects rent. Any account that drops to zero lamports is purged. Accounts can also be markedrent-exempt if they contain a sufficient number of lamports.

In the same way that a Linux user uses a path to look up a file, a Solana client uses anaddress to look up an account. The address is a 256-bit public key.

Signers#

Transactions include one or more digitalsignatures each corresponding to an account address referenced by the transaction. Each of these addresses must be the public key of an ed25519 keypair, and the signature signifies that the holder of the matching private key signed, and thus, "authorized" the transaction. In this case, the account is referred to as asigner . Whether an account is a signer or not is communicated to the program as part of the account's metadata. Programs can then use that information to make authority decisions.

Read-only#

Transactions canindicate that some of the accounts it references be treated asread-only accounts in order to enable parallel account processing between transactions. The runtime permits read-only accounts to be read concurrently by multiple programs. If a program attempts to modify a read-only account, the transaction is rejected by the runtime.

Executable#

If an account is marked "executable" in its metadata, then it is considered a program which can be executed by including the account's public key in an instruction'sprogram id . Accounts are marked as executable during a successful program deployment process by the loader that owns the account. When a program is deployed to the execution engine (SBF deployment), the loader determines that the bytecode in the account's data is valid. If so, the loader permanently marks the program account as executable.

If a program is marked as final (non-upgradeable), the runtime enforces that the account's data (the program) is immutable. Through the upgradeable loader, it is possible to upload a totally new program to an existing program address.

Creating#

To create an account, a client generates akeypair and registers its public key using theSystemProgram::CreateAccount instruction with a fixed storage size in bytes preallocated. The current maximum size of an account's data is 10 MiB, which can be changed (increased or decreased) at a rate over all accounts of 20 MiB per transaction, and the size can be increased by 10 KiB per account and per instruction.

An account address can be any arbitrary 256 bit value, and there are mechanisms for advanced users to create derived addresses (SystemProgram::CreateAccountWithSeed ,Pubkey::CreateProgramAddress).

Accounts that have never been created via the system program can also be passed to programs. When an instruction references an account that hasn't been previously created, the program will be passed an account with no data and zero lamports that is owned by the system program.

Such newly created accounts reflect whether they sign the transaction, and therefore, can be used as an authority. Authorities in this context convey to the program that the holder of the private key associated with the account's public key signed the transaction. The account's public key may be known to the program or recorded in another account, signifying some kind of ownership or authority over an asset or operation the program controls or performs.

Ownership and Assignment to Programs#

A created account is initialized to beowned by a built-in program called the System program and is called asystem account aptly. An account includes "owner" metadata. The owner is a program id. The runtime grants the program write access to the account if its id matches the owner. For the case of the System program, the runtime allows clients to transfer lamports

and importantlyassign account ownership, meaning changing the owner to a different program id. If an account is not owned by a program, the program is only permitted to read its data and credit the account.

Verifying validity of unmodified, reference-only accounts#

For security purposes, it is recommended that programs check the validity of any account it reads, but does not modify.

This is because a malicious user could create accounts with arbitrary data and then pass these accounts to the program in place of valid accounts. The arbitrary data could be crafted in a way that leads to unexpected or harmful program behavior.

The security model enforces that an account's data can only be modified by the account'sOwner program. This allows the program to trust that the data is passed to them via accounts they own. The runtime enforces this by rejecting any transaction containing a program that attempts to write to an account it does not own.

If a program were to not check account validity, it might read an account it thinks it owns, but doesn't. Anyone can issue instructions to a program, and the runtime does not know that those accounts are expected to be owned by the program.

To check an account's validity, the program should either check the account's address against a known value, or check that the account is indeed owned correctly (usually owned by the program itself).

One example is when programs use a sysvar account. Unless the program checks the account's address or owner, it's impossible to be sure whether it's a real and valid sysvar account merely by successful deserialization of the account's data.

Accordingly, the Solana SDKchecks the sysvar account's validity during deserialization. An alternative and safer way to read a sysvar is via the sysvar'sget() function which doesn't require these checks.

If the program always modifies the account in question, the address/owner check isn't required because modifying an unowned account will be rejected by the runtime, and the containing transaction will be thrown out.

Rent#

Keeping accounts alive on Solana incurs a storage cost calledrent because the blockchain cluster must actively maintain the data to process any future transactions. This is different from Bitcoin and Ethereum, where storing accounts doesn't incur any costs.

Currently, all new accounts are required to be rent-exempt.

Rent exemption#

An account is considered rent-exempt if it holds at least 2 years worth of rent. This is checked every time an account's balance is reduced, and transactions that would reduce the balance to below the minimum amount will fail.

Program executable accounts are required by the runtime to be rent-exempt to avoid being purged.

Note: Use thegetMinimumBalanceForRentExemption RPC endpoint to calculate the minimum balance for a particular account size. The following calculation is illustrative only. For example, a program executable with the size of 15,000 bytes requires a balance of 105,290,880 lamports (=~ 0.105 SOL) to be rent-exempt:

105,290,880 = 19.055441478439427 (fee rate) * (128 + 15_000)(account size including metadata) * ((365.25/2) * 2)(epochs in 2 years) Rent can also be estimated via thesolana rent CLI subcommand

solana rent 15000 Rent per byte-year: 0.00000348 SOL Rent per epoch: 0.000288276 SOL Rent-exempt minimum: 0.10529088 SOL Note: Rest assured that, should the storage rent rate need to be increased at some point in the future, steps will be taken to ensure that accounts that are rent-exempt before the increase will remain rent-exempt afterwards