Add Solana to Your Exchange

This guide describes how to add Solana's native token SOL to your cryptocurrency exchange.

Node Setup#

We highly recommend setting up at least two nodes on high-grade computers/cloud instances, upgrading to newer versions promptly, and keeping an eye on service operations with a bundled monitoring tool.

This setup enables you:

- to have a self-administered gateway to the Solana mainnet-beta cluster to get
- data and submit withdrawal transactions
- to have full control over how much historical block data is retained
- to maintain your service availability even if one node fails

Solana nodes demand relatively high computing power to handle our fast blocks and high TPS. For specific requirements, please seehardware recommendations .

To run an api node:

1. Install the Solana command-line tool suite
2. Start the validator with at least the following parameters:

solana-validator \ --ledger \ --identity \ --entrypoint \ --expected-genesis-hash \ --rpc-port 8899 \ --no-voting \ --enable-rpc-transaction-history \ --limit-ledger-size \ --known-validator \ --only-known-rpc Customize--ledger to your desired ledger storage location, and--rpc-port to the port you want to expose.

The--entrypoint and--expected-genesis-hash parameters are all specific to the cluster you are joiningCurrent parameters for Mainnet Beta

The--limit-ledger-size parameter allows you to specify how many ledgershreds your node retains on disk. If you do not include this parameter, the validator will keep the entire ledger until it runs out of disk space. The default value attempts to keep the ledger disk usage under 500GB. More or less disk usage may be requested by adding an argument to--limit-ledger-size if desired. Checksolana-validator --help for the default limit value used by--limit-ledger-size . More information about selecting a custom limit value isavailable here .

Specifying one or more--known-validator parameters can protect you from booting from a malicious snapshotMore on the value of booting with known validators

Optional parameters to consider:

- --private-rpc
- prevents your RPC port from being published for use by other
- nodes
- --rpc-bind-address
- allows you to specify a different IP address to bind the
- RPC port

Automatic Restarts and Monitoring#

We recommend configuring each of your nodes to restart automatically on exit, to ensure you miss as little data as possible. Running the solana software as a systemd service is one great option.

For monitoring, we providesolana-watchtower , which can monitor your validator and detect with thesolana-validator process is unhealthy. It can directly be configured to alert you via Slack, Telegram, Discord, or Twillio. For details, runsolana-watchtower --help .

solana-watchtower --validator-identity You can find more information about thebest practices for Solana Watchtower here in the docs.

New Software Release Announcements#

We release new software frequently (around 1 release / week). Sometimes newer versions include incompatible protocol changes, which necessitate timely software update to avoid errors in processing blocks.

Our official release announcements for all kinds of releases (normal and security) are communicated via adiscord channel called#mb-announcement (mb stands formainnet-beta).

Like staked validators, we expect any exchange-operated validators to be updated at your earliest convenience within a business day or two after a normal release announcement. For security-related releases, more urgent action may be needed.

Ledger Continuity#

By default, each of your nodes will boot from a snapshot provided by one of your known validators. This snapshot reflects the current state of the chain, but does not contain the complete historical ledger. If one of your node exits and boots from a new snapshot, there may be a gap in the ledger on that node. In order to prevent this issue, add the--no-snapshot-fetch parameter to yoursolana-validator command to receive historical ledger data instead of a snapshot.

Do not pass the--no-snapshot-fetch parameter on your initial boot as it's not possible to boot the node all the way from the genesis block. Instead boot from a snapshot first and then add the--no-snapshot-fetch parameter for reboots.

It is important to note that the amount of historical ledger available to your nodes from the rest of the network is limited at any point in time. Once operational if your validators experience significant downtime they may not be able to catch up to the network and will need to download a new snapshot from a known validator. In doing so your validators will now have a gap in its historical ledger data that cannot be filled.

Minimizing Validator Port Exposure#

The validator requires that various UDP and TCP ports be open for inbound traffic from all other Solana validators. While this is the most efficient mode of operation, and is strongly recommended, it is possible to restrict the validator to only require inbound traffic from one other Solana validator.

First add the--restricted-repair-only-mode argument. This will cause the validator to operate in a restricted mode where it will not receive pushes from the rest of the validators, and instead will need to continually poll other validators for blocks. The validator will only transmit UDP packets to other validators using theGossip andServeR ("serve repair") ports, and only receive UDP packets on itsGossip andRepair ports.

TheGossip port is bi-directional and allows your validator to remain in contact with the rest of the cluster. Your validator transmits on theServeR to make repair requests to obtaining new blocks from the rest of the network, since Turbine is now disabled. Your validator will then receive repair responses on theRepair port from other validators.

To further restrict the validator to only requesting blocks from one or more validators, first determine the identity pubkey for that validator and add the--gossip-pull-validator PUBKEY --repair-validator PUBKEY arguments for each PUBKEY. This will cause your validator to be a resource drain on each validator that you add, so please do this sparingly and only after consulting with the target validator.

Your validator should now only be communicating with the explicitly listed validators and only on theGossip ,Repair andServeR ports.

Setting up Deposit Accounts#

Solana accounts do not require any on-chain initialization; once they contain some SOL, they exist. To set up a deposit account for your exchange, simply generate a Solana keypair using any of ourwallet tools .

We recommend using a unique deposit account for each of your users.

Solana accounts must be made rent-exempt by containing 2-years worth ofent in SOL. In order to find the minimum rent-exempt balance for your deposit accounts, query thegetMinimumBalanceForRentExemption endpoint :

```
curl localhost:8899 -X POST -H "Content-Type: application/json" -d '{ "jsonrpc": "2.0", "id": 1, "method":
"getMinimumBalanceForRentExemption", "params":[0] }'
```

Result

```
{ "jsonrpc": "2.0", "result": 890880, "id": 1 }
```

Offline Accounts#

You may wish to keep the keys for one or more collection accounts offline for greater security. If so, you will need to move SOL to hot accounts using ouroffline methods .

Listening for Deposits#

When a user wants to deposit SOL into your exchange, instruct them to send a transfer to the appropriate deposit address.

Versioned Transaction Migration#

When the Mainnet Beta network starts processing versioned transactions, exchangesMUST make changes. If no changes are made, deposit detection will no longer work properly because fetching a versioned transaction or a block containing versioned transactions will return an error.

- {"maxSupportedTransactionVersion": 0}
- ThemaxSupportedTransactionVersion
- parameter must be added togetBlock
- andgetTransaction
- requests to avoid disruption to deposit detection. The latest
- transaction version is0
- and should be specified as the max supported
- transaction version value.

It's important to understand that versioned transactions allow users to create transactions that use another set of account keys loaded from on-chain address lookup tables.

- {"encoding": "jsonParsed"}
- When fetching blocks and transactions, it's now recommended to use the"jsonParsed"
- encoding because it includes all transaction account keys
- (including those from lookup tables) in the message"accountKeys"
- list. This
- makes it straightforward to resolve balance changes detailed inpreBalances
- /postBalances
- andpreTokenBalances
- /postTokenBalances
- .
- If the"json"
- encoding is used instead, entries inpreBalances
- /postBalances
- andpreTokenBalances
- /postTokenBalances
- may refer to
- account keys that areNOT
- in the"accountKeys"
- list and need to be
- resolved using"loadedAddresses"
- entries in the transaction metadata.

Poll for Blocks#

To track all the deposit accounts for your exchange, poll for each confirmed block and inspect for addresses of interest, using the JSON-RPC service of your Solana API node.

- To identify which blocks are available, send agetBlocks
- request, passing the last block
- you have already processed as the start-slot parameter:

```
curl https://api.devnet.solana.com -X POST -H "Content-Type: application/json" -d '{ "jsonrpc": "2.0", "id": 1, "method": "getBlocks", "params": [160017005, 160017015] }'
```

Result

```
{ "jsonrpc": "2.0", "result": [ 160017005, 160017006, 160017007, 160017012, 160017013, 160017014, 160017015 ], "id": 1 }
```

Not every slot produces a block, so there may be gaps in the sequence of integers.

- For each block, request its contents with agetBlock
- request:

Block Fetching Tips#

- {"rewards": false}

By default, fetched blocks will return information about validator fees on each block and staking rewards on epoch boundaries. If you don't need this information, disable it with the "rewards" parameter.

- {"transactionDetails": "accounts"}

By default, fetched blocks will return a lot of transaction info and metadata that isn't necessary for tracking account balances. Set the "transactionDetails" parameter to speed up block fetching.

```
curl https://api.devnet.solana.com -X POST -H 'Content-Type: application/json' -d '{ "jsonrpc": "2.0", "id": 1, "method":
"getBlock", "params": [ 166974442, { "encoding": "jsonParsed", "maxSupportedTransactionVersion": 0, "transactionDetails":
"accounts", "rewards": false } ] }'
```

Result

```
{ "jsonrpc": "2.0", "result": { "blockHeight": 157201607, "blockTime": 1665070281, "blockhash":
"HKhao674uvFc4wMK1Cm3UyuuGbKExdgPFjXQ5xtvsG3o", "parentSlot": 166974441, "previousBlockhash":
"98CNLU4rsYa2HDUyp7PubU4DhwYJJhSX9v6pvE7SWsAo", "transactions": [ ... (omit) { "meta": { "err": null, "fee": 5000,
"postBalances": [ 1110663066, 1, 1040000000 ], "postTokenBalances": [], "preBalances": [ 1120668066, 1, 1030000000 ],
"preTokenBalances": [], "status": { "Ok": null } }, "transaction": { "accountKeys": [ { "pubkey":
"9aE476sH92Vz7DMPyq5WLPkrKWivxeuTKEFKd2sZZcde", "signer": true, "source": "transaction", "writable": true }, {
"pubkey": "11111111111111111111111111111111", "signer": false, "source": "transaction", "writable": false }, { "pubkey":
"G1wZ113tiUHdSpQEBcid8n1x8BAvcWZoZgxPKxgE5B7o", "signer": false, "source": "lookupTable", "writable": true } ],
"signatures": [
"2CxNRsyRT7y88GBwvAB3hRg8wijMSZh3VNYXAdUesGSyvbRJbRR2q9G1KSEpQENmXHmmMLHiXumw4dp8CvzQMjrM"
] }, "version": 0 }, ... (omit) ] }, "id": 1 }
```

ThepreBalances andpostBalances fields allow you to track the balance changes in every account without having to parse the entire transaction. They list the starting and ending balances of each account inlamports , indexed to theaccountKeys list. For example, if the deposit address of interest isG1wZ113tiUHdSpQEBcid8n1x8BAvcWZoZgxPKxgE5B7o , this transaction represents a transfer of 1040000000 - 1030000000 = 10,000,000 lamports = 0.01 SOL

If you need more information about the transaction type or other specifics, you can request the block from RPC in binary format, and parse it using either ourRust SDK orJavascript SDK .

Address History#

You can also query the transaction history of a specific address. This is generallynot a viable method for tracking all your deposit addresses over all slots, but may be useful for examining a few accounts for a specific period of time.

- Send agetSignaturesForAddress
- request to the api node:

```
curl localhost:8899 -X POST -H "Content-Type: application/json" -d '{ "jsonrpc": "2.0", "id": 1, "method":
"getSignaturesForAddress", "params": [ "3M2b3tLji7rvscqrLAHMukYxDK2nB96Q9hwfV6QkdzBN", { "limit": 3 } ] }'
```

Result

```
{ "jsonrpc": "2.0", "result": [ { "blockTime": 1662064640, "confirmationStatus": "finalized", "err": null, "memo": null, "signature":
"3EDRvnD5TbbMS2mCusop6oyHLD8CgnjncaYQd5RXpgnjYUXRCYwiNPmXb6ZG5KdTK4zAaygEhfdLoP7TDzwKBVQp",
"slot": 148697216 }, { "blockTime": 1662064434, "confirmationStatus": "finalized", "err": null, "memo": null, "signature":
"4rPQ5wthgSP1kLdLqcRgQnkYkPAZqjv5vm59LijrQDSKuL2HLmZHoHjdSLDXXWFwWdaKXUuryRBGwEvSxn3TQckY",
"slot": 148696843 }, { "blockTime": 1662064341, "confirmationStatus": "finalized", "err": null, "memo": null, "signature":
"36Q383JMiqiobuPV9qBqy41xjMsVnQBm9rdZSdpbrLTGhSQDTGZJnocM4TQTVfUGfV2vEX9ZB3sex6wUBUWzjEvs",
"slot": 148696677 } ], "id": 1 }
```

* For each signature returned, get the transaction details by sending agetTransaction * request:

```
curl https://api.devnet.solana.com -X POST -H 'Content-Type: application/json' -d '{ "jsonrpc":"2.0", "id":1,
"method":"getTransaction", "params":[
"2CxNRsyRT7y88GBwvAB3hRg8wijMSZh3VNYXAdUesGSyvbRJbRR2q9G1KSEpQENmXHmmMLHiXumw4dp8CvzQMjrM",
{ "encoding":"jsonParsed", "maxSupportedTransactionVersion":0 } ] }'
```

Result

```
{ "jsonrpc": "2.0", "result": { "blockTime": 1665070281, "meta": { "err": null, "fee": 5000, "innerInstructions": [], "logMessages": [
"Program 11111111111111111111111111111111 invoke [1]", "Program 11111111111111111111111111111111 success" ],
"postBalances": [1110663066, 1, 1040000000], "postTokenBalances": [], "preBalances": [1120668066, 1, 1030000000],
"preTokenBalances": [], "rewards": [], "status": { "Ok": null } }, "slot": 166974442, "transaction": { "message": { "accountKeys":
[ { "pubkey": "9aE476sH92Vz7DMPyq5WLPkrKWivxeuTKEFKd2sZZcde", "signer": true, "source": "transaction", "writable":
true }, { "pubkey": "11111111111111111111111111111111", "signer": false, "source": "transaction", "writable": false }, {
"pubkey": "G1wZ113tiUHdSpQEBcid8n1x8BAvcWZoZgxPKxgE5B7o", "signer": false, "source": "lookupTable", "writable":
true } ], "addressTableLookups": [ { "accountKey": "4syr5pBaboZy4cZyF6sys82uGD7jEvoAP2ZMaoich4fZ",
"readonlyIndexes": [], "writableIndexes": [3] } ], "instructions": [ { "parsed": { "info": { "destination":
"G1wZ113tiUHdSpQEBcid8n1x8BAvcWZoZgxPKxgE5B7o", "lamports": 10000000, "source":
"9aE476sH92Vz7DMPyq5WLPkrKWivxeuTKEFKd2sZZcde" }, "type": "transfer" }, "program": "system", "programId":
"11111111111111111111111111111111" } ], "recentBlockhash": "BhhivDNgoy4L5tLtHb1s3TP19uUXqKiy4FfUR34d93eT" },
"signatures": [
```

"2CxNRsyRT7y88GBwvAB3hRg8wijMSZh3VNYXAdUesGSyvbRJbRR2q9G1KSEpQENmXHmmMLHiXumw4dp8CvzQMjrM"] }, "version": 0 }, "id": 1 }

Sending Withdrawals#

To accommodate a user's request to withdraw SOL, you must generate a Solana transfer transaction, and send it to the api node to be forwarded to your cluster.

Synchronous#

Sending a synchronous transfer to the Solana cluster allows you to easily ensure that a transfer is successful and finalized by the cluster.

Solana's command-line tool offers a simple command,solana transfer , to generate, submit, and confirm transfer transactions. By default, this method will wait and track progress on stderr until the transaction has been finalized by the cluster. If the transaction fails, it will report any transaction errors.

solana transfer --allow-unfunded-recipient --keypair --url http://localhost:8899 TheSolana Javascript SDK offers a similar approach for the JS ecosystem. Use theSystemProgram to build a transfer transaction, and submit it using thesendAndConfirmTransaction method.

Asynchronous#

For greater flexibility, you can submit withdrawal transfers asynchronously. In these cases, it is your responsibility to verify that the transaction succeeded and was finalized by the cluster.

Note: Each transaction contains arecent blockhash to indicate its liveness. It iscritical to wait until this blockhash expires before retrying a withdrawal transfer that does not appear to have been confirmed or finalized by the cluster. Otherwise, you risk a double spend. See more onblockhash expiration below.

First, get a recent blockhash using thegetFees endpoint or the CLI command:

solana fees --url http://localhost:8899 In the command-line tool, pass the--no-wait argument to send a transfer asynchronously, and include your recent blockhash with the--blockhash argument:

solana transfer --no-wait --allow-unfunded-recipient --blockhash --keypair --url http://localhost:8899 You can also build, sign, and serialize the transaction manually, and fire it off to the cluster using the JSON-RPCsendTransaction endpoint.

Transaction Confirmations & Finality#

Get the status of a batch of transactions using thegetSignatureStatuses JSON-RPC endpoint. Theconfirmations field reports how manyconfirmed blocks have elapsed since the transaction was processed. Ifconfirmations: null , it isfinalized .

```
curl localhost:8899 -X POST -H "Content-Type: application/json" -d '{ "jsonrpc":"2.0", "id":1,
"method":"getSignatureStatuses", "params":[ [
"5VERv8NMvzbJMEkV8xnrLkEaWRtSz9CosKDYjCJjBRnbJLgp8uirBgmQpjKhoR4tjF3ZpRzrFmBV6UjKdiSZkQUW",
"5j7s6NiJS3JAkvgkoc18WVAsiSaci2pxB2A6ueCJP4tprA2TFg9wSyTLeYouxPBJEMzJinENTkpA52YStRW5Dia7" ] ] }'
```

Result

{ "jsonrpc": "2.0", "result": { "context": { "slot": 82 }, "value": [{ "slot": 72, "confirmations": 10, "err": null, "status": { "Ok": null } }, { "slot": 48, "confirmations": null, "err": null, "status": { "Ok": null } }] }, "id": 1 }

Blockhash Expiration#

You can check whether a particular blockhash is still valid by sending agetFeeCalculatorForBlockhash request with the blockhash as a parameter. If the response value isnull , the blockhash is expired, and the withdrawal transaction using that blockhash should never succeed.

Validating User-supplied Account Addresses for Withdrawals#

As withdrawals are irreversible, it may be a good practice to validate a user-supplied account address before authorizing a withdrawal in order to prevent accidental loss of user funds.

Basic verification#

Solana addresses a 32-byte array, encoded with the bitcoin base58 alphabet. This results in an ASCII text string matching the following regular expression:

[1-9A-HJ-NP-Za-km-z]{32,44} This check is insufficient on its own as Solana addresses are not checksummed, so typos cannot be detected. To further validate the user's input, the string can be decoded and the resulting byte array's length confirmed to be 32. However, there are some addresses that can decode to 32 bytes despite a typo such as a single missing character, reversed characters and ignored case

Advanced verification#

Due to the vulnerability to typos described above, it is recommended that the balance be queried for candidate withdraw addresses and the user prompted to confirm their intentions if a non-zero balance is discovered.

Valid ed25519 pubkey check#

The address of a normal account in Solana is a Base58-encoded string of a 256-bit ed25519 public key. Not all bit patterns are valid public keys for the ed25519 curve, so it is possible to ensure user-supplied account addresses are at least correct ed25519 public keys.

Java#

Here is a Java example of validating a user-supplied address as a valid ed25519 public key:

The following code sample assumes you're using the Maven.

pom.xml :

... spring https://repo.spring.io/libs-release/

...

... io.github.novacrypto Base58 0.1.3 cafe.cryptography curve25519-elisabeth 0.1.0 import io.github.novacrypto.base58.Base58; import cafe.cryptography.curve25519.CompressedEdwardsY;

public class PubkeyValidator { public static boolean verifyPubkey(String userProvidedPubkey) { try { return _verifyPubkeyInternal(userProvidedPubkey); } catch (Exception e) { return false; } }

```
public static boolean _verifyPubkeyInternal(String maybePubkey) throws Exception
{
    byte[] bytes = Base58.base58Decode(maybePubkey);
    return !(new CompressedEdwardsY(bytes)).decompress().isSmallOrder();
}
```

}

Minimum Deposit & Withdrawal Amounts#

Every deposit and withdrawal of SOL must be greater or equal to the minimum rent-exempt balance for the account at the wallet address (a basic SOL account holding no data), currently: 0.000890880 SOL

Similarly, every deposit account must contain at least this balance.

curl localhost:8899 -X POST -H "Content-Type: application/json" -d '{ "jsonrpc": "2.0", "id": 1, "method": "getMinimumBalanceForRentExemption", "params": [0] }'

Result

{ "jsonrpc": "2.0", "result": 890880, "id": 1 }

Prioritization Fees and Compute Units#

In periods of high demand, it’s possible for a transaction to expire before a validator has included such transactions in their block because they chose other transactions with higher economic value. Valid Transactions on Solana may be delayed or dropped if Prioritization Fees are not implemented properly.

Prioritization Fees are additional fees that can be added on top of the base Transaction Fee to ensure transaction inclusion within blocks and in these situations and help ensure deliverability.

These priority fees are added to transaction by adding a special Compute Budget instruction that sets the desired priority fee to be paid.

Important Note Failure to implement these instructions may result in network disruptions and dropped transactions. It is strongly recommended that every exchange supporting Solana make use of priority fees to avoid disruption.

What is a Prioritization Fee?#

Prioritization Fees are priced in micro-lamports per Compute Unit (e.g. small amounts of SOL) prepended to transactions to make them economically compelling for validator nodes to include within blocks on the network.

How much should the Prioritization Fee be?#

The method for setting your prioritization fee should involve querying recent prioritization fees to set a fee which is likely to be compelling for the network. Using thegetRecentPrioritizationFees RPC method, you can query for the prioritization fees required to land a transaction in a recent block.

Pricing strategy for these priority fees will vary based on your use case. There is no canonical way to do so. One strategy for setting your Prioritization Fees might be to calculate your transaction success rate and then increase your Prioritization Fee against a query to the recent transaction fees API and adjust accordingly. Pricing for Prioritization Fees will be dynamic based on the activity on the network and bids placed by other participants, only knowable after the fact.

One challenge with using thegetRecentPrioritizationFees API call is that it may only return the lowest fee for each block. This will often be zero, which is not a fully useful approximation of what Prioritization Fee to use in order to avoid being rejected by validator nodes.

ThegetRecentPrioritizationFees API takes accounts' pubkeys as parameters, and then returns the highest of the minimum prioritization fees for these accounts. When no account is specified, the API will return the lowest fee to land to block, which is usually zero (unless the block is full).

Exchanges and applications should query the RPC endpoint with the accounts that a transaction is going to write-lock. The RPC endpoint will return themax(account_1_min_fee, account_2_min_fee, ... account_n_min_fee) , which should be the base point for the user to set the prioritization fee for that transaction.

There are different approaches to setting Prioritization Fees and somethird-party APIs are available to determine the best fee to apply. Given the dynamic nature of the network, there will not be a "perfect" way to go about pricing your Prioritization fees and careful analysis should be applied before choosing a path forward.

How to Implement Prioritization Fees#

Adding priority fees on a transaction consists of prepending two Compute Budget instructions on a given transaction:

- one to set the compute unit price, and
- another to set the compute unit limit

Info Here, you can also find a more detailed developerguide on how to use priority fees which includes more information about implementing priority fees. Create asetComputeUnitPrice instruction to add a Prioritization Fee above the Base Transaction Fee (5,000 Lamports).

// import { ComputeBudgetProgram } from "@solana/web3.js" ComputeBudgetProgram.setComputeUnitPrice({ microLamports: number }); The value provided in micro-lamports will be multiplied by the Compute Unit (CU) budget to determine the Prioritization Fee in Lamports. For example, if your CU budget is 1M CU, and you add1 microLamport/CU , the Prioritization Fee will be 1 lamport (1M * 0. 000001). The total fee will then be 5001 lamports.

To set a new compute unit budget for the transaction, create asetComputeUnitLimit instruction

// import { ComputeBudgetProgram } from "@solana/web3.js" ComputeBudgetProgram.setComputeUnitLimit({ units: number }); Theunits value provided will replace the Solana runtime's default compute budget value.

Set the lowest CU required for the transaction Transactions should request the minimum amount of compute units (CU) required for execution to maximize throughput and minimize overall fees.

You can get the CU consumed by a transaction by sending the transaction on a different Solana cluster, like devnet. For example, asimple token transfer takes 300 CU. // import { ... } from "@solana/web3.js"

const modifyComputeUnits = ComputeBudgetProgram.setComputeUnitLimit({ // note: set this to be the lowest actual CU consumed by the transaction units: 300, });

const addPriorityFee = ComputeBudgetProgram.setComputeUnitPrice({ microLamports: 1, });

const transaction = new Transaction() .add(modifyComputeUnits) .add(addPriorityFee) .add(SystemProgram.transfer({ fromPubkey: payer.publicKey, toPubkey: toAccount, lamports: 10000000, }),);

Prioritization Fees And Durable Nonces#

If your setup uses Durable Nonce Transactions, it is important to properly implement Prioritization Fees in combination with

Durable Transaction Nonces to ensure successful transactions. Failure to do so will cause intended Durable Nonce transactions not to be detected as such.

If you ARE using Durable Transaction Nonces, theAdvanceNonceAccount instruction MUST be specified FIRST in the instructions list, even when the compute budget instructions are used to specify priority fees.

You can find a specific code exampleusing durable nonces and priority fees together in this developer guide.

Supporting the SPL Token Standard#

SPL Token is the standard for wrapped/synthetic token creation and exchange on the Solana blockchain.

The SPL Token workflow is similar to that of native SOL tokens, but there are a few differences which will be discussed in this section.

Token Mints#

Eachtype of SPL Token is declared by creating amint account. This account stores metadata describing token features like the supply, number of decimals, and various authorities with control over the mint. Each SPL Token account references its associated mint and may only interact with SPL Tokens of that type.

Installing the spl-token CLI Tool#

SPL Token accounts are queried and modified using thespl-token command line utility. The examples provided in this section depend upon having it installed on the local system.

spl-token is distributed from Rustcrates.io via the Rustcargo command line utility. The latest version ofcargo can be installed using a handy one-liner for your platform atrustup.rs . Oncecargo is installed,spl-token can be obtained with the following command:

cargo install spl-token-cli You can then check the installed version to verify

spl-token --version Which should result in something like

spl-token-cli 2.0.1

Account Creation#

SPL Token accounts carry additional requirements that native System Program accounts do not:

1. SPL Token accounts must be created before an amount of tokens can be
2. deposited. Token accounts can be created explicitly with thespl-token create-account
3. command, or implicitly by thespl-token transfer --fund-recipient ...
4. command.
5. SPL Token accounts must remainrent-exempt
6. for the duration of
7. their existence and therefore require a small amount of native SOL tokens be
8. deposited at account creation. For SPL Token accounts, this amount is
9. 0.00203928 SOL (2,039,280 lamports).

Command Line#

To create an SPL Token account with the following properties:

1. Associated with the given mint
2. Owned by the funding account's keypair

spl-token create-account

Example#

spl-token create-account AkUFCWTXb3w9nY2n6SFJvBV6VwvFUCe4KBMCcgLsa2ir Creating account 6VzWGL51jLebvnDifvcuEDec17sK6Wupi4gYhm5RzfkV Signature: 4JsqZEPra2eDTHtHpB4FMWSfk3UgcCVmkKkP7zESZeMrKmFFkDkNd91pKP3vPVVZZPiu5XxyJwS73Vi5WsZL88D7 Or to create an SPL Token account with a specific keypair:

solana-keygen new -o token-account.json spl-token create-account AkUFCWTXb3w9nY2n6SFJvBV6VwvFUCe4KBMCcgLsa2ir token-account.json Creating account 6VzWGL51jLebvnDifvcuEDec17sK6Wupi4gYhm5RzfkV Signature:

4JsqZEPra2eDTHtHpB4FMWSfk3UgcCVmkKkP7zESZeMrKmFFkDkNd91pKP3vPVVZZPiu5XxyJwS73Vi5WsZL88D7

Checking an Account's Balance#

Command Line#

spl-token balance

Example#

solana balance 6VzWGL51jLebvnDifvcuEDec17sK6Wupi4gYhm5RzfkV 0

Token Transfers#

The source account for a transfer is the actual token account that contains the amount.

The recipient address however can be a normal wallet account. If an associated token account for the given mint does not yet exist for that wallet, the transfer will create it provided that the--fund-recipient argument as provided.

Command Line#

spl-token transfer --fund-recipient

Example#

spl-token transfer 6B199xxzw3PkAm25hGJpjj3Wj3WNYNHzDAnt1tEqg5BN 1 6VzWGL51jLebvnDifvcuEDec17sK6Wupi4gYhm5RzfkV Transfer 1 tokens Sender: 6B199xxzw3PkAm25hGJpjj3Wj3WNYNHzDAnt1tEqg5BN Recipient: 6VzWGL51jLebvnDifvcuEDec17sK6Wupi4gYhm5RzfkV Signature: 3R6tsog17QM8KfzbcbdP4aoMfwgo6hBggJDVy7dZPVmH2xbCWjEj31JKD53NzMrf25ChFjY7Uv2dfCDq4mGFFyAj

Depositing#

Since each(wallet, mint) pair requires a separate account on chain. It is recommended that the addresses for these accounts be derived from SOL deposit wallets using theAssociated Token Account (ATA) scheme and thatonly deposits from ATA addresses be accepted.

Monitoring for deposit transactions should follow theblock polling method described above. Each new block should be scanned for successful transactions referencing user token-account derived addresses. ThepreTokenBalance andpostTokenBalance fields from the transaction's metadata must then be used to determine the effective balance change. These fields will identify the token mint and account owner (main wallet address) of the affected account.

Note that if a receiving account is created during the transaction, it will have nopreTokenBalance entry as there is no existing account state. In this case, the initial balance can be assumed to be zero.

Withdrawing#

The withdrawal address a user provides must be that of their SOL wallet.

Before executing a withdrawaltransfer , the exchange should check the address asdescribed above . Additionally this address must be owned by the System Program and have no account data. If the address has no SOL balance, user confirmation should be obtained before proceeding with the withdrawal. All other withdrawal addresses must be rejected.

From the withdrawal address, theAssociated Token Account (ATA) for the correct mint is derived and the transfer issued to that account via aTransferChecked instruction. Note that it is possible that the ATA address does not yet exist, at which point the exchange should fund the account on behalf of the user. For SPL Token accounts, funding the withdrawal account will require 0.00203928 SOL (2,039,280 lamports).

Templatespl-token transfer command for a withdrawal:

spl-token transfer --fund-recipient

Other Considerations#

Freeze Authority#

For regulatory compliance reasons, an SPL Token issuing entity may optionally choose to hold "Freeze Authority" over all accounts created in association with its mint. This allows them tofreeze the assets in a given account at will, rendering the account unusable until thawed. If this feature is in use, the freeze authority's pubkey will be registered in the SPL Token's mint account.

Basic Support for the SPL Token-2022 (Token-Extensions) Standard#

SPL Token-2022 is the newest standard for wrapped/synthetic token creation and exchange on the Solana blockchain.

Also known as "Token Extensions", the standard contains many new features that token creators and account holders may optionally enable. These features include confidential transfers, fees on transfer, closing mints, metadata, permanent delegates, immutable ownership, and much more. Please see theextension guide for more information.

If your exchange supports SPL Token, there isn't a lot more work required to support SPL Token-2022:

- the CLI tool works seamlessly with both programs starting with version 3.0.0.
- preTokenBalances
- andpostTokenBalances
- include SPL Token-2022 balances
- RPC indexes SPL Token-2022 accounts, but they must be queried separately with
- program idTokenzQdBNbLqP5VEhdkAS6EPFLC1PHnBqCXEpPxuEb

The Associated Token Account works the same way, and properly calculates the required deposit amount of SOL for the new account.

Because of extensions, however, accounts may be larger than 165 bytes, so they may require more than 0.00203928 SOL to fund.

For example, the Associated Token Account program always includes the "immutable owner" extension, so accounts take a minimum of 170 bytes, which requires 0.00207408 SOL.

Extension-Specific Considerations#

The previous section outlines the most basic support for SPL Token-2022. Since the extensions modify the behavior of tokens, exchanges may need to change how they handle tokens.

It is possible to see all extensions on a mint or token account:

spl-token display

Transfer Fee#

A token may be configured with a transfer fee, where a portion of transferred tokens are withheld at the destination for future collection.

If your exchange transfers these tokens, beware that they may not all arrive at the destination due to the withheld amount.

It is possible to specify the expected fee during a transfer to avoid any surprises:

spl-token transfer --expected-fee --fund-recipient

Mint Close Authority#

With this extension, a token creator may close a mint, provided the supply of tokens is zero.

When a mint is closed, there may still be empty token accounts in existence, and they will no longer be associated to a valid mint.

It is safe to simply close these token accounts:

spl-token close --address

Confidential Transfer#

Mints may be configured for confidential transfers, so that token amounts are encrypted, but the account owners are still public.

Exchanges may configure token accounts to send and receive confidential transfers, to hide user amounts. It is not required to enable confidential transfers on token accounts, so exchanges can force users to send tokens non-confidentially.

To enable confidential transfers, the account must be configured for it:

spl-token configure-confidential-transfer-account --address And to transfer:

spl-token transfer --confidential During a confidential transfer, thepreTokenBalance andpostTokenBalance fields will show no change. In order to sweep deposit accounts, you must decrypt the new balance to withdraw the tokens:

spl-token apply-pending-balance --address spl-token withdraw-confidential-tokens --address

Default Account State#

Mints may be configured with a default account state, such that all new token accounts are frozen by default. These token creators may require users to go through a separate process to thaw the account.

Non-Transferable#

Some tokens are non-transferable, but they may still be burned and the account can be closed.

Permanent Delegate#

Token creators may designate a permanent delegate for all of their tokens. The permanent delegate may transfer or burn tokens from any account, potentially stealing funds.

This is a legal requirement for stablecoins in certain jurisdictions, or could be used for token repossession schemes.

Beware that these tokens may be transferred without your exchange's knowledge.

Transfer Hook#

Tokens may be configured with an additional program that must be called during transfers, in order to validate the transfer or perform any other logic.

Since the Solana runtime requires all accounts to be explicitly passed to a program, and transfer hooks require additional accounts, the exchange needs to create transfer instructions differently for these tokens.

The CLI and instruction creators such ascreateTransferCheckedWithTransferHookInstruction add the extra accounts automatically, but the additional accounts may also be specified explicitly:

spl-token transfer --transfer-hook-account --transfer-hook-account ...

Required Memo on Transfer#

Users may configure their token accounts to require a memo on transfer.

Exchanges may need to prepend a memo instruction before transferring tokens back to users, or they may require users to prepend a memo instruction before sending to the exchange:

spl-token transfer --with-memo

Testing the Integration#

Be sure to test your complete workflow on Solana devnet and testnetclusters before moving to production on mainnet-beta. Devnet is the most open and flexible, and ideal for initial development, while testnet offers more realistic cluster configuration. Both devnet and testnet support a faucet, runsolana airdrop 1 to obtain some devnet or testnet SOL for development and testing.