Versioned Transactions

Versioned Transactions are the new transaction format that allow for additional functionality in the Solana runtime, includingAddress Lookup Tables .

While changes toon chain programs areNOT required to support the new functionality of versioned transactions (or for backwards compatibility), developersWILL need update their client side code to preventerrors due to different transaction versions .

Current Transaction Versions#

The Solana runtime supports two transaction versions:

- legacy
- older transaction format with no additional benefit
- 0
- added support forAddress Lookup Tables

Max supported transaction version#

All RPC requests that return a transactionshould specify the highest version of transactions they will support in their application using themaxSupportedTransactionVersion option, includinggetBlock andgetTransaction .

An RPC request will fail if aVersioned Transaction is returned that is higher than the setmaxSupportedTransactionVersion . (i.e. if a version0 transaction is returned whenlegacy is selected)

WARNING: If nomaxSupportedTransactionVersion value is set, then onlylegacy transactions will be allowed in the RPC response. Therefore, your RPC requestsWILL fail if any version0 transactions are returned.

How to set max supported version#

You can set themaxSupportedTransactionVersion using both the@solana/web3.js library and JSON formatted requests directly to an RPC endpoint.

Using web3.js#

Using the@solana/web3.js library, you can retrieve the most recent block or get a specific transaction:

// connect to the devnet cluster and get the current slot const connection = new web3.Connection(web3.clusterApiUrl("devnet")); const slot = await connection.getSlot();

// get the latest block (allowing for v0 transactions) const block = await connection.getBlock(slot, { maxSupportedTransactionVersion: 0, });

// get a specific transaction (allowing for v0 transactions) const getTx = await connection.getTransaction("3jpoANiFeVGisWRY5UP648xRXs3iQasCHABPWRWnoEjeA93nc79WrnGgpgazjq4K9m8g2NJoyKoWBV1Kx5VmtwHQ", { maxSupportedTransactionVersion: 0, },);

JSON requests to the RPC#

Using a standard JSON formatted POST request, you can set themaxSupportedTransactionVersion when retrieving a specific block:

curl http://localhost:8899 -X POST -H "Content-Type: application/json" -d \ '{"jsonrpc": "2.0", "id":1, "method": "getBlock", "params": [430, { "encoding":"json", "maxSupportedTransactionVersion":0, "transactionDetails":"full", "rewards":false }]}'

How to create a Versioned Transaction#

Versioned transactions can be created similar to the older method of creating transactions. There are differences in using certain libraries that should be noted.

Below is an example of how to create a Versioned Transaction, using the@solana/web3.js library, to send perform a SOL transfer between two accounts.

Notes:#

- payer
- is a validKeypair
- wallet, funded with SOL
- toAccount
- a validKeypair

Firstly, import the web3.js library and create aconnection to your desired cluster.

We then define the recentblockhash andminRent we will need for our transaction and the account:

const web3 = require("@solana/web3.js");

// connect to the cluster and get the minimum rent for rent exempt status const connection = new web3.Connection(web3.clusterApiUrl("devnet")); let minRent = await connection.getMinimumBalanceForRentExemption(0); let blockhash = await connection .getLatestBlockhash() .then(res => res.blockhash); Create anarray of all theinstructions you desire to send in your transaction. In this example below, we are creating a simple SOL transfer instruction:

// create an array with your desiredinstructions const instructions = [web3.SystemProgram.transfer({ fromPubkey: payer.publicKey, toPubkey: toAccount.publicKey, lamports: minRent, }),]; Next, construct aMessageV0 formatted transaction message with your desiredinstructions :

// create v0 compatible message const messageV0 = new web3.TransactionMessage({ payerKey: payer.publicKey, recentBlockhash: blockhash, instructions, }).compileToV0Message(); Then, create a newVersionedTransaction , passing in our v0 compatible message:

const transaction = new web3.VersionedTransaction(messageV0);

// sign your transaction with the requiredSigners transaction.sign([payer]); You can sign the transaction by either:

- passing an array ofsignatures
- into theVersionedTransaction
- method, or
- call thetransaction.sign()
- method, passing an array of the requiredSigners

NOTE: After calling thetransaction.sign() method, all the previous transactionsignatures will be fully replaced by new signatures created from the provided inSigners . After yourVersionedTransaction has been signed by all required accounts, you can send it to the cluster andawait the response:

// send our v0 transaction to the cluster const txid = await connection.sendTransaction(transaction); console.log(https://explorer.solana.com/tx{txid}?cluster=devnet); NOTE: Unlikelegacy transactions, sending aVersionedTransaction viasendTransaction doesNOT support transaction signing via passing in an array ofSigners as the second parameter. You will need to sign the transaction before callingconnection.sendTransaction() .

More Resources#

- usingVersioned Transactions for Address Lookup Tables
- view anexample of a v0 transaction
- on Solana Explorer
- read theaccepted proposal
- for Versioned Transaction and Address Lookup Tables