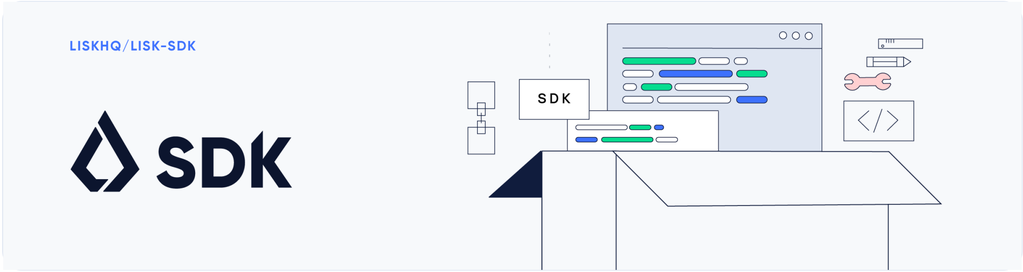
Lisk SDK



The Lisk SDK is an open-source software development kit which enables developers to build scalable blockchain applications in JavaScript.

Highlights

The most important highlights and features of the Lisk SDK:

- **One blockchain per app**: blockchain applications each run on their own customized blockchain.
- Programming language: **(JavaScript) &** **(TypeScript)**
- Consensus algorithm: **Lisk DPoS** & **Lisk BFT**
- Completely modular: Configure, exchange or extend any of the blockchain related logic in the application with **Modules and assets** and **Plugins**.
- **Convenient development tools**:
  - Lisk Commander:
    - Bootstrap a complete default blockchain application with a single command.
    - Generate skeletons for modules, assets, and plugins.
    - Use the Lisk SDK libraries directly in the terminal via the interactive Lisk console.
    - Many other convenient CLI tools such as creating accounts, encrypting passphrases / messages and more.
  - Dashboard plugin: Communicate conveniently with your blockchain application during development via a dashboard.
  - SDK testing utilities: simplifies the testing of all components of a blockchain application by providing dedicated fixtures, mocks, and utility functions.
- **Interoperable** (coming soon, see roadmap): Make your app interoperable with other blockchain applications, by registering it as a sidechain on the Lisk mainchain.

Quickstart

The quickest way to bootstrap a blockchain application with the Lisk SDK is explained here.

Table of Contents

Supported Operating Systems

Dependencies

- Node.js
- Lisk Commander

Project setup

Start the application

Using the application CLI globally

Supported Operating Systems

The Lisk SDK will run on both the Ubuntu and MacOS operating systems listed below:

Ubuntu | MacOS

- 18.04 (LTS)
- 20.04 (LTS)

Note

Please note that the SDK does not come with any official support for Windows.

Dependencies

The following dependencies need to be installed to run the Lisk SDK:

Dependencies	Version
Git	v2 (latest)
Node.js	v16.15.0
Lisk Commander	v5.1.10 (latest)

Node.js

If you are using NVM, (Node.js Version Manager), ensure you install the correct version as shown below:

```
nvm install v16.15.0
```

Lisk Commander

It is recommended to install Lisk Commander globally with NPM (Node Package Manager), to facilitate the convenient usage of the Lisk Commander CLI.

```
npm install --global lisk-commander
```

Project setup

```
mkdir new_app
cd new_app
```

new_app/

```
lisk init
```

The user is asked for the application name, description, author, and license after running `lisk init`.

Next, all the required files are created by Lisk Commander.

```
Using template "lisk-ts"
Initializing git repository
Updating .liskrc.json file
Creating project structure
? Application name new_app
? Application description My blockchain application
? Author mona
? License ISC
```

Start the application

Execute the following command to start the blockchain application.

new_app/

```
./bin/run start
```

To verify that the application has started, check the log messages in the terminal. If the start was successful, the application will enable forging for all genesis delegates and will begin adding new blocks to the blockchain every 10 seconds.

Note

The application CLI offers various commands to conveniently manage your blockchain application.

To see an overview of all the available CLI commands, execute the following commands in the root folder of the application:

new_app/

```
./bin/run --help
```

Using the application CLI globally

To use the application commands globally, create an alias in the `.bashrc` file.

~/.bashrc/

```
alias new_app="$HOME/new_app/bin/run" (1)
```

(1) Adjust to the path where your `new_app` is located.

After updating the `.bashrc` file, make it directly available in the terminal by executing the following command:

```
. ~/.bashrc
```

Now it is possible to conveniently run the application CLI commands from anywhere by referring to the alias.

```
new_app --help
```

Application CLI

A complete command reference of the application CLI is summarized on this page.

Try it live

Full operations CLI is only available on L2 Nitrox, if the blockchain application has [Advanced gas cost functionality](#).

Table of Contents

- Commands
  - help
  - version
  - evmdata
  - evmtrace
  - evmdebugtrace
- Topics
  - accounts
    - accounts.newaccount
    - accounts.getbal
    - accounts.sendtx
  - blockchain
    - blockchain.getblock
    - blockchain.importchain
    - blockchain.exportchain
    - blockchain.insert
  - config
    - config.dump
    - config.get
  - debug
    - debug.getblockrlp
    - debug.setblock
  - gasless
    - gasless.getgaslessstate
    - gasless.getgaslessstats
    - gasless.getgaslesswhitelist
    - gasless.getgaslessblacklist
    - gasless.getgaslessgasprice
  - governance
    - governance.getvotes
    - governance.getvotingpower
    - governance.getquorum
    - governance.getproposal
    - governance.getproposalstate
    - governance.getproposalthreshold

Commands

help

Shows the general help information for the application CLI.

```
NAME:
   application - A new cmd application built with Cosmos SDK v0.47.0 and Ethermint

USAGE:
   application [global options] command [command options] [arguments...]

VERSION:
   0.1.0

COMMANDS:
   config      Create and interact with configuration files
   debug       Tool for debugging data structures
   gasless     Gasless transaction management commands
   governance  Governance proposal and voting commands
   keys        Manage your application's keys
   query       Querying subcommands
   tx          Transactions subcommands
   help, h     Shows a list of commands or help for one command

GLOBAL OPTIONS:
   --chain-id value  Chain ID of tendermint node
   --node value      <host>:<port> to tendermint rpc interface for this chain (default: "tcp://localhost:26657")
   --grpc-addr value  the gRPC endpoint to use for this chain
   --grpc-insecure   allow gRPC over insecure channels, if TLS the server must use TLS
   --help, -h        show help
   --version, -v     print the version
```

version

Shows the blockchain node.

```
NAME:
   application version - Print application version

USAGE:
   application version [command options] [arguments...]

DESCRIPTION:
   The output of this command depends on the application configuration.
   You can set the application version in the app.toml or config.toml
   configuration files which is set during compilation time by passing
   the application version using ldflags in the form of

   -X github.com/cosmos/cosmos-sdk/version.AppName=simd -X github.com/cosmos/cosmos-sdk/version.Version=1.0 -X github.com/cosmos/cosmos-sdk/version.Commit=f0f7b7dab7e36c20b757cebce0e8f4fc5b95de60

   See the file scripts/ldflags.sh

OPTIONS:
   --long, -l  Print extra information
   --help, -h  show help
```

evmdata

Shows the EVM, available for use commands.

```
NAME:
   application evmdata - Query and convert with EVM compatibility

USAGE:
   application evmdata [command options] [arguments...]

COMMANDS:
   tx-hash  Query for a transaction by Ethereum hash
   help, h  Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

evmtracetransaction

Displays executable transaction instructions.

```
NAME:
   application evmtrace - Trace a transaction

USAGE:
   application evmtrace [command options] [arguments...]

DESCRIPTION:
   Trace a transaction by its hash and return the execution trace.
   Supports various tracer types including callTracer, 4byteTracer, and prestateTracer.

   Examples:
     application evmtrace 0x1234...abcd
     application evmtrace 0x1234...abcd --tracer callTracer
     application evmtrace 0x1234...abcd --tracer 4byteTracer --timeout 30s

OPTIONS:
   --tracer value, -t value   Tracer type: callTracer, 4byteTracer, prestateTracer, or empty for default struct logger
   --timeout value            Trace timeout (e.g., 5s, 30s, 1m) (default: 5s)
   --disable-memory           Disable memory capture in trace
   --disable-stack            Disable stack capture in trace
   --disable-storage          Disable storage capture in trace
   --enable-return-data       Enable return data capture in trace
   --help, -h                 show help
```

evmdebugtrace

Shows executable transaction instructions.

```
NAME:
   application evmdebugtrace - Debug trace a transaction with detailed execution info

USAGE:
   application evmdebugtrace [command options] [arguments...]

DESCRIPTION:
   Debug trace a transaction by its hash and return detailed execution information.
   This is a convenience command that defaults to the structLogger tracer.

   Examples:
     application evmdebugtrace 0x1234...abcd
     application evmdebugtrace 0x1234...abcd --timeout 60s

OPTIONS:
   --timeout value            Trace timeout (e.g., 5s, 30s, 1m) (default: 5s)
   --disable-memory           Disable memory capture in trace
   --disable-stack            Disable stack capture in trace
   --disable-storage          Disable storage capture in trace
   --help, -h                 show help
```

Topics

accounts

Commands relating to Ethereum accounts.

```
NAME:
   application accounts - Manage Ethereum accounts

USAGE:
   application accounts [command options] [arguments...]

COMMANDS:
   list        Print a list of known accounts
   new         Create a new account. Enter a password when prompted
   update      Update an account. Enter a password when prompted
   import      Import a private key into a new account
   help, h     Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

accounts.newaccount

Returns a randomly generated Ethereum private key with its corresponding address(public key) and can add address.

```
NAME:
   application accounts new - Create a new account

USAGE:
   application accounts new [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

accounts.getbal

Shows an account's balance.

```
NAME:
   application accounts list - List all accounts

USAGE:
   application accounts list [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

accounts.sendtx

Sends a transaction to an address.

```
NAME:
   application accounts update - Update the passphrase of an account

USAGE:
   application accounts update [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

accounts.importkey

Imports a private key into a new account.

```
NAME:
   application accounts import - Import a private key into a new account

USAGE:
   application accounts import [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

blockchain

Commands relating to the blockchain data.

```
NAME:
   application blockchain - Blockchain subcommands

USAGE:
   application blockchain [command options] [arguments...]

COMMANDS:
   block       Get block by height
   blocks      Get blocks by range
   validators  Get validators by height
   help, h     Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

blockchain.getblock

Gets the blockchain block.

```
NAME:
   application blockchain block - Get block by height

USAGE:
   application blockchain block [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

blockchain.importchain

Imports the blockchain data from file.

```
NAME:
   application blockchain import - Import blocks from file

USAGE:
   application blockchain import [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

blockchain.exportchain

Exports the blockchain data to a file.

```
NAME:
   application blockchain export - Export blocks to file

USAGE:
   application blockchain export [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

blockchain.insert

Inserts RAW/RLP hash from the blockchain data.

```
NAME:
   application blockchain insert - Insert blocks from RLP

USAGE:
   application blockchain insert [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

blockchain.import

Imports the blockchain data from file.

```
NAME:
   application blockchain import - Import blocks from file

USAGE:
   application blockchain import [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

blockchain.reset

Resets the blockchain data.

```
NAME:
   application blockchain reset - Reset to genesis or specific block

USAGE:
   application blockchain reset [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

config

Commands relating to the configuration of the blockchain application.

```
NAME:
   application config - Application configuration

USAGE:
   application config [command options] [arguments...]

COMMANDS:
   dump        Dump configuration
   get         Get configuration value
   help, h     Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

config.dump

Creates a configuration file for the blockchain application.

```
NAME:
   application config dump - Dump configuration

USAGE:
   application config dump [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

config.get

Shows the application config.

```
NAME:
   application config get - Get configuration value

USAGE:
   application config get [command options] [arguments...]

DESCRIPTION:
   Get application configuration values without opening the config file.

   Examples:
     application config get server.minimum-gas-prices
     application config get api.enable
     application config get grpc.address

OPTIONS:
   --help, -h  show help
```

debug

Commands relating to the Ethereum application.

```
NAME:
   application debug - Debug commands

USAGE:
   application debug [command options] [arguments...]

COMMANDS:
   raw         Print raw RLP of any base64, hex, or bech32 value
   addr        Convert an address between hex and bech32
   pubkey      Decode a pubkey from hex, base64, or bech32
   help, h     Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

debug.getblockrlp

Prints the RLP encoding of the block.

```
NAME:
   application debug block - Print block info

USAGE:
   application debug block [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

debug.setblock

Sets the head block hash.

```
NAME:
   application debug setblock - Set block

USAGE:
   application debug setblock [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

gasless

Commands relating to gasless transaction management.

```
NAME:
   application gasless - Gasless transaction management commands

USAGE:
   application gasless [command options] [arguments...]

COMMANDS:
   getgaslessstate      Get gasless transaction state
   getgaslessstats      Get gasless transaction statistics
   getgaslesswhitelist  Get gasless whitelist
   getgaslessblacklist  Get gasless blacklist
   getgaslessgasprice   Get gasless gas price
   help, h              Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

gasless.getgaslessstate

Shows the current gasless transaction state.

```
NAME:
   application gasless getgaslessstate - Get gasless transaction state

USAGE:
   application gasless getgaslessstate [command options] [arguments...]

DESCRIPTION:
   Get the current state of gasless transaction processing.

   Example:
     application gasless getgaslessstate

OPTIONS:
   --help, -h  show help
```

gasless.getgaslessstats

Shows gasless transaction statistics.

```
NAME:
   application gasless getgaslessstats - Get gasless transaction statistics

USAGE:
   application gasless getgaslessstats [command options] [arguments...]

DESCRIPTION:
   Get statistics about gasless transaction processing.

   Example:
     application gasless getgaslessstats

OPTIONS:
   --help, -h  show help
```

gasless.getgaslesswhitelist

Shows the gasless whitelist.

```
NAME:
   application gasless getgaslesswhitelist - Get gasless whitelist

USAGE:
   application gasless getgaslesswhitelist [command options] [arguments...]

DESCRIPTION:
   Get the list of addresses that are whitelisted for gasless transactions.

   Example:
     application gasless getgaslesswhitelist

OPTIONS:
   --help, -h  show help
```

gasless.getgaslessblacklist

Shows the gasless blacklist.

```
NAME:
   application gasless getgaslessblacklist - Get gasless blacklist

USAGE:
   application gasless getgaslessblacklist [command options] [arguments...]

DESCRIPTION:
   Get the list of addresses that are blacklisted for gasless transactions.

   Example:
     application gasless getgaslessblacklist

OPTIONS:
   --help, -h  show help
```

gasless.getgaslessgasprice

Shows the gasless gas price.

```
NAME:
   application gasless getgaslessgasprice - Get gasless gas price

USAGE:
   application gasless getgaslessgasprice [command options] [arguments...]

DESCRIPTION:
   Get the current gas price for gasless transactions.

   Example:
     application gasless getgaslessgasprice

OPTIONS:
   --help, -h  show help
```

governance

Governance proposal and voting commands

```
NAME:
   application governance - Governance proposal and voting commands

USAGE:
   application governance [command options] [arguments...]

COMMANDS:
   getvotes            Get votes for a proposal
   getvotingpower      Get voting power for an address
   getquorum           Get quorum requirement
   getproposal         Get proposal details
   getproposalstate    Get proposal state
   getproposalthreshold Get proposal threshold
   help, h             Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

governance.getvotes

Shows votes for a proposal.

```
NAME:
   application governance getvotes - Get votes for a proposal

USAGE:
   application governance getvotes [command options] [arguments...]

DESCRIPTION:
   Get all votes cast for a specific proposal.

   Example:
     application governance getvotes 1

OPTIONS:
   --limit value   Limit number of votes returned (default: 100)
   --offset value  Offset for pagination (default: 0)
   --help, -h      show help
```

governance.getvotingpower

Shows voting power for an address.

```
NAME:
   application governance getvotingpower - Get voting power for an address

USAGE:
   application governance getvotingpower [command options] [arguments...]

DESCRIPTION:
   Get the voting power for a specific address.

   Example:
     application governance getvotingpower 0x1234...abcd

OPTIONS:
   --help, -h  show help
```

governance.getquorum

Shows the quorum requirement.

```
NAME:
   application governance getquorum - Get quorum requirement

USAGE:
   application governance getquorum [command options] [arguments...]

DESCRIPTION:
   Get the quorum requirement for governance proposals.

   Example:
     application governance getquorum

OPTIONS:
   --help, -h  show help
```

governance.getproposal

Shows proposal details.

```
NAME:
   application governance getproposal - Get proposal details

USAGE:
   application governance getproposal [command options] [arguments...]

DESCRIPTION:
   Get detailed information about a specific proposal.

   Example:
     application governance getproposal 1

OPTIONS:
   --help, -h  show help
```

governance.getproposalstate

Shows proposal state.

```
NAME:
   application governance getproposalstate - Get proposal state

USAGE:
   application governance getproposalstate [command options] [arguments...]

DESCRIPTION:
   Get the current state of a proposal (pending, active, succeeded, defeated, etc.).

   Example:
     application governance getproposalstate 1

OPTIONS:
   --help, -h  show help
```

governance.getproposalthreshold

Shows proposal threshold.

```
NAME:
   application governance getproposalthreshold - Get proposal threshold

USAGE:
   application governance getproposalthreshold [command options] [arguments...]

DESCRIPTION:
   Get the minimum voting power required to create a proposal.

   Example:
     application governance getproposalthreshold

OPTIONS:
   --help, -h  show help
```

keys

Manage your application's keys.

```
NAME:
   application keys - Manage your application's keys

USAGE:
   application keys [command options] [arguments...]

DESCRIPTION:
   Keyring management commands. These keys may be in any format supported by the
   Tendermint crypto library and can be used by light-clients, full nodes, or any other application
   that needs to sign with a private key.

   The keyring supports the following backends:

       os          Uses the operating system's default credentials store.
       file        Uses encrypted file-based keystore within the app's configuration directory.
                   This keyring will request a password each time it is accessed.
       kwallet     Uses KDE Wallet Manager as a credentials management application.
       pass        Uses the pass command line utility to store and retrieve keys.
       test        Stores keys insecurely to disk. It does not prompt for a password to be unlocked
                   and it should be used only for testing purposes.

   kwallet and pass backends depend on external tools. Refer to their respective documentation for more
   information:
       KWallet     https://github.com/KDE/kwallet
       pass        https://www.passwordstore.org/

   The pass backend requires GnuPG: https://gnupg.org/
   In addition, pass requires a proper GPG key setup. The gpg agent
   must be available in order to pass work properly.

   It is recommended to set the backend explicitly by using the --keyring-backend flag.

   The pass backend requires a keyring directory to be specified. It can be set via the
   --keyring-dir flag or by setting the directory in client.toml.

   The test backend is a testing keyring that uses the OS file system and does not encrypt keys.
   Never use it in production.

   The os backend relies on OS-specific defaults. For instance, macOS Keychain and Windows Credentials
   Manager. On Linux it relies on the Linux kernel keyring.

   The file backend stores keys in a directory specified via the --keyring-dir flag.
   It is the most cross-platform backend, and suitable for server environments.

   The kwallet backend stores keys in a dedicated KDE Wallet application. On systems where a dbus session
   is available, the backend will attempt to open the default wallet. It is possible to select a custom
   wallet with the --keyring-dir flag.

   The pass backend relies on keys that were added to pass in order to function correctly. Keys added to
   pass will be prefixed by "cosmos-app-" and should be listed as such.

   The test backend stores keys in a directory specified via the --keyring-dir flag.
   It is intended for testing purposes and should not be used in production environments.

   The memory backend stores keys in memory. They will be lost after the process exits.
   It is intended for testing purposes and should not be used in production environments.

COMMANDS:
   add                  Add an encrypted private key (either newly generated or imported) or a key stored in a hardware wallet to the keyring
   delete               Delete the given key
   export               Export private keys
   import               Import private keys into the local keybase
   list                 List all keys
   migrate              Migrate keys from amino to proto serialization
   mnemonic             Compute the bip39 mnemonic for some input entropy
   parse                Parse address from hex to bech32 and vice versa
   rename               Rename the given key
   show                 Retrieve key information by name or address
   help, h              Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

keys.add

Add an encrypted private key.

```
NAME:
   application keys add - Add an encrypted private key (either newly generated or imported) or a key stored in a hardware wallet to the keyring

USAGE:
   application keys add [command options] [arguments...]

DESCRIPTION:
   Derive a new private key and encrypt to disk.
   Optionally specify a BIP39 mnemonic, a BIP39 passphrase to further secure the mnemonic,
   and a bip32 HD path to derive a specific account. The key will be stored under the given name
   and encrypted with the given password. The only input that is required is the password.

   The default number of words (24) may be overwritten with the --words flag, but carries
   no security if entropy is not provided via the --recover flag.

   The --pubkey flag can be used to create a public key only key entry.

   Entries to the ledger or offline storage can be added by setting the --ledger or --offline
   flags, or by setting the --hw-wallet flag to the desired hardware wallet device.

   Use the --dry-run flag to print the address and public key without saving the key.

   Use the --interactive flag to be prompted for the bip39 passphrase and mnemonic.

   Use the --multisig-threshold flag to create a multisig key from a set of keys.

   Use the --no-sort flag to disable sorting the multisig keys.

   Use the --nosave flag to print the key_name, mnemonic, and address without saving the key.

   Use the --sign-mode flag to create a key with a specific sign mode (direct or amino-json).

   Use the --key-algorithm flag to specify the algorithm to use when creating the key.

   Use the --output flag to specify the output format (text, json).

   Use the --coin-type flag to specify a coin type.

   Use the --key-type flag to specify a key type.

   Use the --account flag to specify an account number.

   Use the --index flag to specify an index number.

   Use the --offline flag to specify an offline key.

   Use the --ledger flag to specify a ledger key.

   Use the --hw-wallet flag to specify a hardware wallet key.

   Use the --dry-run flag to print the address and public key without saving the key.

   Use the --interactive flag to be prompted for the bip39 passphrase and mnemonic.

   Use the --multisig-threshold flag to create a multisig key from a set of keys.

   Use the --no-sort flag to disable sorting the multisig keys.

   Use the --nosave flag to print the key_name, mnemonic, and address without saving the key.

   Use the --sign-mode flag to create a key with a specific sign mode (direct or amino-json).

   Use the --key-algorithm flag to specify the algorithm to use when creating the key.

   Use the --output flag to specify the output format (text, json).

   Use the --coin-type flag to specify a coin type.

   Use the --key-type flag to specify a key type.

   Use the --account flag to specify an account number.

   Use the --index flag to specify an index number.

   Use the --offline flag to specify an offline key.

   Use the --ledger flag to specify a ledger key.

   Use the --hw-wallet flag to specify a hardware wallet key.

OPTIONS:
   --help, -h  show help
```

keys.delete

Delete the given key.

```
NAME:
   application keys delete - Delete the given key

USAGE:
   application keys delete [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.export

Export private keys.

```
NAME:
   application keys export - Export private keys

USAGE:
   application keys export [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.import

Import private keys into the local keybase.

```
NAME:
   application keys import - Import private keys into the local keybase

USAGE:
   application keys import [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.list

List all keys.

```
NAME:
   application keys list - List all keys

USAGE:
   application keys list [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.migrate

Migrate keys from amino to proto serialization.

```
NAME:
   application keys migrate - Migrate keys from amino to proto serialization

USAGE:
   application keys migrate [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.mnemonic

Compute the bip39 mnemonic for some input entropy.

```
NAME:
   application keys mnemonic - Compute the bip39 mnemonic for some input entropy

USAGE:
   application keys mnemonic [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.parse

Parse address from hex to bech32 and vice versa.

```
NAME:
   application keys parse - Parse address from hex to bech32 and vice versa

USAGE:
   application keys parse [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.rename

Rename the given key.

```
NAME:
   application keys rename - Rename the given key

USAGE:
   application keys rename [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

keys.show

Retrieve key information by name or address.

```
NAME:
   application keys show - Retrieve key information by name or address

USAGE:
   application keys show [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

query

Querying subcommands.

```
NAME:
   application query - Querying subcommands

USAGE:
   application query [command options] [arguments...]

COMMANDS:
   block       Get block by height
   blocks      Get blocks by range
   validators  Get validators by height
   help, h     Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

query.block

Gets the blockchain block.

```
NAME:
   application query block - Get block by height

USAGE:
   application query block [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

query.blocks

Gets the blockchain blocks.

```
NAME:
   application query blocks - Get blocks by range

USAGE:
   application query blocks [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

query.validators

Gets the validators.

```
NAME:
   application query validators - Get validators by height

USAGE:
   application query validators [command options] [arguments...]

OPTIONS:
   --help, -h  show help
```

tx

Transactions subcommands.

```
NAME:
   application tx - Transactions subcommands

USAGE:
   application tx [command options] [arguments...]

COMMANDS:
   sign        Sign a transaction
   broadcast   Broadcast a transaction
   help, h     Shows a list of commands or help for one command

OPTIONS:
   --help, -h  show help
```

tx.sign

Sign a transaction.

```
NAME:
   application tx sign - Sign a transaction

USAGE:
   application tx sign [command options] [arguments...]

DESCRIPTION:
   Sign a transaction. The transaction must be signed by the key with the given name.

OPTIONS:
   --help, -h  show help
```

tx.broadcast

Broadcast a transaction.

```
NAME:
   application tx broadcast - Broadcast a transaction

USAGE:
   application tx broadcast [command options] [arguments...]

DESCRIPTION:
   Broadcast a transaction to the network.

OPTIONS:
   --help, -h  show help
```

Application Configuration

The application config object is passed to the `Application` instance and must follow the configuration schema.

It is not necessary to include all options that are included in the schema. The application will use the default configuration options in case a config option is not specified.

> **Tip**
>
> For more information about the config schema, see the application config schema file at GitHub.

Default config options

```
const config = {
    // label defines the process name and folder within the root path
        "label": "beta-sdk-app",
        // version must follow semver format
        "version": "0.0.0",
        // network version defines a P2P network version
        "networkVersion": "1.0",
    // rootPath defines the root path for all data to be stored
        "rootPath": "~/.lisk",
        // logger holds information for the logging
        "logger": {
            // fileLogLevel defines the log level output for the file logging
                "fileLogLevel": "info",
                // consoleLogLevel defines the log level output for the console logging
                "consoleLogLevel": "info",
                // logFileName defines a name for the log file
                "logFileName": "lisk.log"
        },
        // rpc defines communication behavior
        "rpc": {
            // enabled creates IPC or WS socket if true
                "enable": false,
                // enabled communication through 'ipc' or 'ws'
                "mode": "ipc",
                // In case of `mode` is set to `ws`, this port used
                "port": 8080,
        // Change to 0.0.0.0 to connect from a remote server
        "host": "127.0.0.1"
        },
        // genesisConfig holds the blockchain protocol configuration
    // it is also passed to the module constructor
        "genesisConfig": {
            // blockTime defines the frequency of blocks to be created
                "blockTime": 10,
                // communityIdentifier defines a community identifier used to create the network identifier
                "communityIdentifier": "sdk",
                // maxPayloadLength defines a maximum payload size allowed in a block in bytes
                "maxPayloadLength": 15360,
                // bftThreshold defines a threshold for pre-vote and pre-commit
                "bftThreshold": 68,
                // minFeePerByte defines a minimum fee per byte for a transaction
                "minFeePerByte": 1000,
                // baseFees defines an additional base fee to be included in the calculation of the minimum
                "baseFees": [
                        {
                                "moduleID": 5,
                                "assetID": 0,
                                "baseFee": "1000000000"
                        }
                ],
                // rewards defines a block reward schedule
                "rewards": {
                    // milestones defines the block reward for every distance
                        "milestones": ["500000000", "400000000", "300000000", "200000000", "100000000"],
                        // offset defines at which height the block reward is given
                        "offset": 2160,
                        // distance defines the duration of the each milestone
                        "distance": 3000000
                },
                // The minimum balance of accounts
                "minRemainingBalance": "5000000",
                // Number of actively forging delegates.
                "activeDelegates": 101,
                // Number of random standy delegates that are allowed to forge each round.
                "standbyDelegates": 2,
                 // The offset of rounds from the current round, which will be used to calculate the vote w
                "delegateListRoundOffset": 2
        },
        // forging holds delegate information for forging
        "forging": {
                "force": true,
                // waitThreshold defines the Number of seconds to wait for previous block before forging
                "waitThreshold": 2,
                // delegates holds the delegate information for forging
                "delegates": [
                        {
                            // encryptedPassphrase defines the encrypted passphrase
                                "encryptedPassphrase": "iterations=1000000&cipherText=5c53db41ec94b46049ca5
                                // hashOnion holds the seed reveal to put in block header
                                "hashOnion": {
                                    // count holds the total number of hash onions
                                        "count": 100000,
                                        // distance holds a distance between each hash onion
                                        "distance": 1000,
                                        // hashes holds the seed reveal for every distance
                                        "hashes": [
                                                "34ecc432170c0812e7ca69d73485ca57",
                                                "1bf9423f594619f7d14e6f742c0631a1",
                        // ...
                                                "fa51b75c7920894019b43378af621e2d",
                                                "bd4ea06be86fb6d850023be7ad1d9558",
                                                "da23c5a34d19bbd57ebb159da170dfb5"
                                        ]
                                },
                                // address defines the address of the delegate
                                "address": "68d6b039567ebbfc714176d87cdd6906cf526cc7"
                        }
                ],
                // defaultPassword defines a password to use to decrypt the encrypted Passphrase
                "defaultPassword": "state dawn marriage honey cinnamon sadness crumble someone file caution
        },
        // network holds the network information of the node
        "network": {
            // seedPeers defines an entry point of the network
                "seedPeers": [
                        {
                                "ip": "127.0.0.1",
                                "port": 5000
                        }
                ],
                // port defines an open port for P2P incoming connections
                "port": 5000,
                // (Optional) blacklistedIPs defines IP address which the node will reject the connection f
        //"blacklistedIPs": string[],
        // (Optional) fixedPeers defines peers which will always try to connect for outbound connections
        // Warning! The connectivity of the node might be negatively impacted if using this option.
        // "fixedPeers": { ip: string, port: number }[],
        // (Optional) whitelistedPeers defines peers that are always allowed to connect to the node on inbo
        // Warning! Beware of declaring only trustworthy peers in this array as these could attack a
                // node with a denial-of-service attack because the banning mechanism is deactivated.
        // whitelistedPeers?: { ip: string, port: number }[],
        // (Optional) peerBanTime defines the length of banning in milliseconds
        // Default: 86400000 (24h)
        //"peerBanTime": number,
        // (Optional) connectTimeout defines a timeout for a connection
        // "connectTimeout": number,
        // Optional.
        // (Optional) actTimeout defines a timeout for response from a peer
        //"ackTimeout": number,
        // (Optional) maxOutboundConnections defines a maximum number of outbound connection allowed
        // Default: 20
        //"maxOutboundConnections": number,
        // (Optional) maxInboundConnections defines a maximum number of inbound connection allowed
        // Default: 100
        //"maxInboundConnections": number,
        // (Optional) sendPeerLimit defines a maximum peer to send information when "send" is called
        // Default: 16
        //"sendPeerLimit": number,
        // (Optional) maxPeerDiscoveryResponseLength defines a maximum length for the peer information resp
        // Default: 200
        //"maxPeerDiscoveryResponseLength": number,
        // (Optional) wsMaxPayload defines maximum size of the payload allowed per communication
        // Default: 3048576
        //"wsMaxPayload": number,
        // (Optional) advertiseAddress defines whether to announce the IP/Port other peers
        // Default: true
        //"advertiseAddress": boolean
        },
        // (Optional) transactionPool defines custom properties of the transaction pool
        "transactionPool": {
            // maxTransactions defines a maximum number of transactions in the pool
                "maxTransactions": 4096,
                // maxTransactionsPerAccount defines a maximum number of transactions in the pool per sende
                "maxTransactionsPerAccount": 64,
                // transactionExpiryTime defines timeout of the transaction in the pool in milliseconds
                "transactionExpiryTime": 10800000,
                // minEntranceFeePriority defines a minimum fee priority required to be added to the transa
                "minEntranceFeePriority": "0",
                // minReplacementFeeDifference defines a minimum fee difference to replace a transaction wi
                "minReplacementFeeDifference": "10"
        },
        // (Optional) plugins holds a group of plugin-specific configs which is passed to a particular plug
        "plugins": {
            // Example config for the HTTP API plugin
        /*"httpApi": {
            "port": 4000,
            "host": "127.0.0.1",
            "whiteList": ["127.0.0.1"],
            "cors": {
                "origin": "*",
                "methods": ["GET", "POST", "PUT"],
            },
            "limits": {
                "max": 0,
                "delayMs": 0,
                "delayAfter": 0,
                "windowMs": 60000,
                "headersTimeout": 5000,
                "serverSetTimeout": 20000,
            },
        }*/
        }
}
```

Actions

Actions are invoked to receive specific data from the blockchain application. Actions are part of the public request / response API, and are invoked via RPCs.

Tip

For mor information about the public application endpoints, check out the RPC API for Lisk nodes page.

Table of Contents

Module- and plugin-specific actions

Application actions

- app:getRegisteredActions
- app:getRegisteredEvents
- app:getRegisteredModules
- app:getNodeInfo
- app:getConnectedPeers
- app:getForgingStatus
- app:updateForgingStatus
- app:getAccount
- app:getAccounts
- app:getLastBlock
- app:getBlockByID
- app:getBlocksByIDs
- app:getBlockByHeight
- app:getBlocksByHeightBetween
- app:getTransactionByID
- app:getTransactionsByIDs
- app:getTransactionsFromPool
- app:postTransaction
- app:getForgers
- app:getSchema

Module- and plugin-specific actions

Modules and plugins often expose additional actions to the application. All actions and events registered with the application are available as RPC endpoints.

The actions and events are described on the reference pages for the different modules, see Modules overview and Plugins overview.

Tip

To see a full list of all available actions and events of your blockchain application, invoke the actions `# app:getRegisteredActions` and `# app:getRegisteredModules`.

Application actions

app:getRegisteredActions

Returns a list of all actions known to the application.

Includes actions from registered modules and plugins.

Input

none

Returns

app:getRegisteredEvents

Returns a list of all events known to the application.

Includes events from registered modules and plugins.

Input

none

Returns

app:getRegisteredModules

Gets a list of all modules that are registered in the application.

Input

none

Returns

app:getNodeInfo

Gets information about the node.

Input

none

Returns

app:getConnectedPeers

Returns all connected peers.

Input

none

Returns

app:getForgingStatus

Gets information about the forging status of the node.

Input

none

Returns

app:updateForgingStatus

Enable or disable forging for a registered forger in the config.

Input

```
{ address: string; (1)
  password: string; (2)
  forging: boolean; (3)
  height?: number; (4)
  maxHeightPreviouslyForged?: number; (5)
  maxHeightPrevoted?: number; (6)
  overwrite?: boolean; (7)
}
```

(1) Address as hex string.
(2) Password to decrypt the passphrase.
(3) When enabling forging, the value should be `true`.
(4) Height of the last forged block by the delegate.
(5) Only required when enabling forging. Height of the previously prevoted block by any delegate. Must match the value in the `forger_info` data.
(6) Only required when enabling forging. Height of the previously last forged block. Must match the value in the `forger_info` data.
(7) Optional: If true, overrides `maxHeightPreviouslyForged` and `maxHeightPrevoted` values in the forger_info data.

Returns

app:getAccount

Gets information about an account based on its address.

Example

```
client.invoke('app:getAccount', {
    address: '8ba7f64f59f7c6a6f748027765578b4a7dd55ea6'
}).then(res => {
    console.log(res);
    const accObject = client.account.decode(res);
    const accJSON = client.account.toJSON(accObject);
    console.log(accJSON);
});
```

Input

```
{ address: string; (1)
}
```

(1) Address as hex string.

Returns

app:getAccounts

Gets information about multiple accounts based on their address.

Input

```
{ address: string[]; (1)
}
```

(1) List of addresses in hexadecimal representation.

Returns

app:getLastBlock

Gets the last forged block of the blockchain.

Input

none

Returns

app:getBlockByID

Gets information about a block based on its ID.

Input

```
{ id: string; (1)
}
```

(1) block ID in hex string

Returns

app:getBlocksByIDs

Gets information about multiple blocks based on their IDs.

Input

```
{ ids: string[]; (1)
}
```

(1) block ID in hex string

Returns

app:getBlockByHeight

Gets information about a block based on its height.

Input

```
{ height: number; (1)
}
```

(1) block height

Returns

app:getBlocksByHeightBetween

Gets all blocks starting from height `from`, until height `to`.

Input

```
{ from: number; (1)
  to: number; (2)
}
```

(1) block height to fetch from
(2) block height to fetch to

Returns

app:getTransactionByID

Gets information about a single transaction based on its ID.

Input

```
{ id: string; (1)
}
```

(1) transaction ID in hex string

Returns

app:getTransactionsByIDs

Gets information about multiple transactions based on their IDs.

Input

```
{ ids: string[]; (1)
}
```

(1) transaction ID in hex string

Returns

app:getTransactionsFromPool

Gets information about all transactions that are currently in the transaction pool of the application.

Input

none

Returns

app:postTransaction

Posts a transaction to the node.

Input

```
{ transaction: string; (1)
}
```

(1) encoded transaction in hex string

app:getForgers

Returns the status of all registered forgers information for the current round.

Input

none

Returns

app:getSchema

Gets the app schema.

Input

none

Returns

Events

Events are part of the public publish / subscribe API of a blockchain application.

Tip

For mor information about the public application endpoints, check out the RPC API for Lisk nodes page

Table of Contents

Module- and plugin-specific events

Application events

- app:ready
- app:shutdown
- app:network:ready
- app:network:event
- app:transaction:new
- app:chain:fork
- app:chain:validators:change
- app:block:new
- app:block:delete

Module- and plugin-specific events

Modules and plugins often expose additional events to the application. All actions and events registered with the application are available as RPC endpoints.

The actions and events are described on the reference pages for the different modules, see Modules overview and Plugins overview.

Application events

app:ready

Fired when the application starts.

Returns

app:shutdown

Fired when the application stops.

Returns

app:network:ready

Fired when the network has at least one outbound connection.

app:network:event

Fired when the application receives a P2P event from the network.

Returns

app:transaction:new

Fired when the node receives a new transaction.

Returns

app:chain:fork

Fired when the node received a block from the forked chain.

Returns

app:chain:validators:change

Fired when the node updates the validator set.

Returns

app:block:new

Fired when a new block is added to the blockchain.

Returns

app:block:delete

Emitted when a block is deleted from the blockchain.

Returns

Modules overview

An overview of all available modules for the Lisk SDK.

Name	Module ID	Description
Token module	2	Handles all logic related to balance. Specifically the following criteria: • Validating and subtracting fees for all transactions. • Checking the minimum remaining balance requirement. • Giving block rewards to the block generator. • Transferring account balances.
Sequence module	3	Handles all logic related to the nonce. It should verify the nonce for all transactions and increment it if valid.
Keys module	4	Handles all logic related to the signatures. It is responsible for verifying signatures of a transaction from multi-signature and non-multi-signature accounts. It also handles the registration of multi-signature accounts.
DPoS module	5	Responsible for handling all DPoS related logics. Specifically the following: • Snapshotting vote weights • Calculating productivity • Handling registerDelegate, voteDelegate, unlockToken, and reportDelegateMisbehavior transaction assets • Setting the next delegates set

Token module

The token module handles all logic related to balance. Specifically the following criteria:

- Validating and subtracting fees for all transactions.
- Checking the minimum remaining balance requirement.
- Giving block rewards to the block generator.
- Transferring account balances.

Name	`token`
ID	`2`
Assets	**ID** / **Name**: 0 / TransferAsset
Reducers	`credit()`, `debit()`, `getBalance()`, `getMinRemainingBalance()`
Actions	none
Events	none

Table of Contents

Account schema

Transactions

- TransferAsset

Reducers

- credit()
- debit()
- getBalance()
- getMinRemainingBalance()

Account schema

The token module adds a new property `balance` under the key `token` to every account in the network as follows:

```
{
    type: 'object',
    properties: {
        balance: {
            fieldNumber: 1,
            dataType: 'uint64',
        },
    },
    default: {
        balance: BigInt(0),
    },
}
```

Transactions

The following transaction assets are provided by the token module.

TransferAsset

Allows the possibility to execute a transfer transaction, which transfers tokens from one account to another.

Schema

```
{
    $id: 'lisk/transfer-asset',
    title: 'Transfer transaction asset',
    type: 'object',
    required: ['amount', 'recipientAddress', 'data'],
    properties: {
        amount: {
            dataType: 'uint64',
            fieldNumber: 1,
        },
        recipientAddress: {
            dataType: 'bytes',
            fieldNumber: 2,
            minLength: 20,
            maxLength: 20,
        },
        data: {
            dataType: 'string',
            fieldNumber: 3,
            minLength: 0,
            maxLength: 64,
        },
    },
}
```

Reducers

credit()

Credits a specific amount of tokens to an account.

credit()

```
async (params: Record<string, unknown>, stateStore: StateStore): Promise<void> => {
    const { address, amount } = params;
    if (!Buffer.isBuffer(address)) {
        throw new Error('Address must be a buffer');
    }
    if (typeof amount !== 'bigint') {
        throw new Error('Amount must be a bigint');
    }
    if (amount <= BigInt(0)) {
        throw new Error('Amount must be a positive bigint.');
    }
    const account = await stateStore.account.getOrDefault<TokenAccount>(address);
    account.token.balance += amount;
    if (account.token.balance < this._minRemainingBalance) {
        throw new Error(
            `Remaining balance must be greater than ${this._minRemainingBalance.toString()}`,
        );
    }
    await stateStore.account.set(address, account);
},
```

debit()

Debits a specific amount of tokens from an account.

debit()

```
async (params: Record<string, unknown>, stateStore: StateStore): Promise<void> => {
    const { address, amount } = params;
    if (!Buffer.isBuffer(address)) {
        throw new Error('Address must be a buffer');
    }
    if (typeof amount !== 'bigint') {
        throw new Error('Amount must be a bigint');
    }
    if (amount <= BigInt(0)) {
        throw new Error('Amount must be a positive bigint.');
    }
    const account = await stateStore.account.getOrDefault<TokenAccount>(address);
    account.token.balance -= amount;
    if (account.token.balance < this._minRemainingBalance) {
        throw new Error(
            `Remaining balance must be greater than ${this._minRemainingBalance.toString()}`,
        );
    }
    await stateStore.account.set(address, account);
},
```

getBalance()

Get the balance of an specific account.

getBalance()

```
async (
    params: Record<string, unknown>,
    stateStore: StateStore,
): Promise<bigint> => {
    const { address } = params;
    if (!Buffer.isBuffer(address)) {
        throw new Error('Address must be a buffer');
    }
    const account = await stateStore.account.getOrDefault<TokenAccount>(address);
    return account.token.balance;
},
```

getMinRemainingBalance()

Returns the minimum remaining balance for accounts.

getMinRemainingBalance()

```
async (): Promise<bigint> => this._minRemainingBalance,
```

Sequence module

The Sequence module handles all logic related to the nonce.

It should verify the nonce for all transactions and increment it if valid.

Name	`sequence`
ID	`3`
Assets	none
Reducers	none
Actions	none
Events	none

Account schema

The sequence module adds a new property `nonce` under the key `sequence` to every account in the network as follows:

```
{
    type: 'object',
    properties: {
        nonce: {
            fieldNumber: 1,
            dataType: 'uint64',
        },
    },
    default: {
        nonce: BigInt(0),
    },
}
```

Keys module

The Keys module handles all logic related to the signatures.

It is responsible for verifying signatures of a transaction from multi-signature and non-multi-signature accounts. It also handles the registration of multi-signature accounts.

Name	`keys`
ID	`4`
Assets	**ID** / **Name**: 0 — RegisterAsset
Reducers	none
Actions	none
Events	none

Table of Contents

Account schema

Transactions

- RegisterAsset

Account schema

The keys module adds the new properties `mandatoryKeys`, `optionalKeys`, and `numberOfSignatures` under the key `keys` to every account in the network as follows:

```
{
    type: 'object',
    properties: {
        numberOfSignatures: { dataType: 'uint32', fieldNumber: 1 },
        mandatoryKeys: {
            type: 'array',
            items: { dataType: 'bytes' },
            fieldNumber: 2,
        },
        optionalKeys: {
            type: 'array',
            items: { dataType: 'bytes' },
            fieldNumber: 3,
        },
    },
    default: {
        mandatoryKeys: [],
        optionalKeys: [],
        numberOfSignatures: 0,
    },
}
```

Transactions

The following transaction assets are provided by the keys module.

RegisterAsset

Allows to send a `registerMultisignatureGroup` transaction, which creates a multi-signature account.

Schema

```
{
        $id: 'lisk/keys/register',
        type: 'object',
        required: ['numberOfSignatures', 'optionalKeys', 'mandatoryKeys'],
        properties: {
                numberOfSignatures: {
                        dataType: 'uint32',
                        fieldNumber: 1,
                        minimum: 1,
                        maximum: 64,
                },
                mandatoryKeys: {
                        type: 'array',
                        items: {
                                dataType: 'bytes',
                                minLength: 32,
                                maxLength: 32,
                        },
                        fieldNumber: 2,
                        minItems: 0,
                        maxItems: 64,
                },
                optionalKeys: {
                        type: 'array',
                        items: {
                                dataType: 'bytes',
                                minLength: 32,
                                maxLength: 32,
                        },
                        fieldNumber: 3,
                        minItems: 0,
                        maxItems: 64,
                },
        },
}
```

DPoS module

The DPoS module is responsible for handling all DPoS related logics. Specifically the following:

1. Snapshotting vote weights
2. Calculating productivity
3. Handling registerDelegate, voteDelegate, unlockToken, and reportDelegateMisbehavior transaction assets
4. Setting the next delegates set

Name	dpos
ID	5

Assets	ID	Value
	0	Register TransactionAsset
	1	VoteTransactionAsset
	2	UnlockTransactionAsset
	3	PomTransactionAsset

Reducers	none
Actions	• 0: dposgetAllDelegates • 1: dposgetAllVotedlogs

Events	none
Custom config options	Option	Default value
	activeDelegates	101
	standbyDelegates	2
	delegateListRoundOffset	2

Table of Contents

Account schema
Transactions
- Register TransactionAsset
- Vote TransactionAsset
- Unlock TransactionAsset
- Pom TransactionAsset

Actions
- dposgetAllDelegates
- dposgetAllVotedlogs

Delegated Proof of Stake (DPoS)
- Forging
- Registering a delegate
- Voting for a delegate
- Unvoting delegates and unlocking of tokens

BFT
- How to discover misbehavior of a delegate
- Reporting a delegate
- Consequences for punished delegates and their voters

Account schema

The token module adds a new property, `dposVotes`, under the key `token` in every account in the network as follows:

```
{
  "dpos": {
    "delegate": {
      "username": {
        "type": "string",
        "fieldNumber": 1,
        "minLength": 1,
        "maxLength": 20
      },
      "pomHeights": {
        "type": "array",
        "items": {
          "dataType": "uint32"
        },
        "fieldNumber": 2
      },
      "consecutiveMissedBlocks": {
        "dataType": "uint32",
        "fieldNumber": 3
      },
      "lastForgedHeight": {
        "dataType": "uint32",
        "fieldNumber": 4
      },
      "isPunished": {
        "dataType": "boolean",
        "fieldNumber": 5
      }
    },
    "sentVotes": {
      "type": "array",
      "fieldNumber": 2,
      "items": {
        "type": "object",
        "properties": {
          "delegateAddress": {
            "dataType": "bytes",
            "fieldNumber": 1
          },
          "amount": {
            "dataType": "uint64",
            "fieldNumber": 2
          }
        }
      }
    },
    "receivedVotes": { "$ref": "sentVotes" },
    "unlocking": {
      "type": "array",
      "fieldNumber": 3,
      "items": {
        "type": "object",
        "properties": {
          "delegateAddress": {
            "dataType": "bytes",
            "fieldNumber": 1
          },
          "amount": {
            "dataType": "uint64",
            "fieldNumber": 2
          },
          "unvoteHeight": {
            "dataType": "uint32",
            "fieldNumber": 3
          }
        }
      }
    }
  },
  "token": {
    "balance": {
      "dataType": "uint64",
      "fieldNumber": 1
    }
  }
}
```

Transactions

The following transaction assets are provided by the token module.

RegisterTransactionAsset

Allows sending a register delegate transaction, which registers a delegate for the sender account with a given username.

Name

`registerDelegate`

ID

0

Fee

10 LSK

Schema

```
{ "type": "object",
  "required": [ "username" ],
  "properties": {
    "username": {
      "dataType": "string",
      "minLength": 1,
      "maxLength": 20,
      "fieldNumber": 1
    }
  }
}
```

VoteTransactionAsset

Allows sending a vote transaction, which casts votes and unvotes for delegates.

Each token can only be used once for voting, therefore the sender locks a certain amount of tokens for each vote. After unvoting a delegate, the user is able to unlock the token again with the `UnlockTransactionAsset`.

Name

`voteDelegate`

ID

1

Schema

```
{ "type": "object",
  "required": [ "votes" ],
  "properties": {
    "votes": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "delegateAddress": { "dataType": "bytes", "fieldNumber": 1 },
          "amount": { "dataType": "sint64", "fieldNumber": 2 }
        }
      }
    }
  }
}
```

UnlockTransactionAsset

Allows sending an unlock transaction, which unlocks tokens that have been locked after voting for a delegate, after waiting for the delay period.

Name

`unlockToken`

ID

2

Schema

```
{ "type": "object",
  "required": [ "unlockObjects" ],
  "properties": {
    "unlockObjects": {
      "type": "array",
      "minItems": 0,
      "maxItems": 20,
      "items": {
        "type": "object",
        "required": [ "delegateAddress", "amount", "unvoteHeight" ],
        "properties": {
          "delegateAddress": { "dataType": "bytes", "fieldNumber": 1 },
          "amount": { "dataType": "uint64", "fieldNumber": 2 },
          "unvoteHeight": { "dataType": "uint32", "fieldNumber": 3 }
        }
      }
    }
  }
}
```

PomTransactionAsset

Allows sending a proof-of-misbehavior transaction, and reports violations of the BFT protocol by a particular delegate.

Name

`reportDelegateMisbehavior`

ID

3

Schema

```
{ "type": "object",
  "required": [ "header1", "header2" ],
  "properties": {
    "header1": {
      "dataType": "bytes",
      "fieldNumber": 1
    },
    "header2": {
      "dataType": "bytes",
      "fieldNumber": 2
    }
  }
}
```

Actions

dposgetAllDelegates

Returns a list of all registered delegates, including their username and address. The address is returned as a hex string.

Input

none

Returns

```
[
  { username: string, },
  { address: string, }
]
```

① Username of the delegate.
② Address of the delegate as hex string.

dposgetAllVotedlogs

Returns a list of delegate unvotes of a certain account, the height of the unvote, and the minimum height for unlocking the tokens again.

Input

```
{ address: string }
```

① Address of the account as hex string.

Returns

```
[
  { delegateAddress: string, },
  { amount: string, },
  { unvoteHeight: number, },
  { canUnlockAtHeight: number, }
]
```

① Address of the delegate as hex string.
② Amount of tokens that were unvoted.
③ Height of the block in which the unvote was forged.
④ Minimum height at which the tokens can be unlocked.

Delegated Proof of Stake (DPoS)

The [LSK DPoS](https://lisk.com/documentation/lisk-sdk/understand-blockchain/consensus/dpos-consensus.html) bootstraps a blockchain network that is based on the [Lisk DPoS](https://lisk.com/documentation/lisk-sdk/understand-blockchain/consensus/dpos-consensus.html) consensus algorithm.

In DPoS based blockchains, the consensus regarding who can forge the next block is reached by users according to the following:

> [!info]
> The DPoS used by Lisk is in fact more of a middle ground between [PoS] and DPoS.
>
> The DPoS newer characteristic is the ability of users to [register as a delegate] and then [receive votes] from other users, in order to become one of the forgers.
>
> The PoS variant characteristic is the requirement for delegates to lock-up a certain amount of tokens, in order to increase their delegate weight. Another PoS characteristic is the mechanism for the selection of the [pseudo-random] algorithm, who are more likely to be selected the higher the delegate weight, the higher the chance to be selected in one of the next seven seeds available for creating [forgers].

Forging

The process of adding new blocks to a blockchain that uses the PoS or DPoS consensus algorithm is called forging.

The 101 [active delegates] and 2 additional [standby delegates] are selected to forge during a forging round.

> [!info]
> During a forging round, no two calculations are required, which makes DPoS an extremely **energy friendly** consensus algorithm, compared to "mining", which is the wasting process for blockchains that run with Proof of Work (PoW). The technique enables a very high number of transactions per second, which means that forging nodes can even run on machines with very limited processing capabilities, such as a Raspberry Pi.

`101(amount of the active delegates) = 3(random standby delegates) + 98(#Number of blocks of a forging round)`

More information about the delegate selection mechanism can be found in the [Lisk Protocol].

Delegate weight

The 101 delegates with the highest delegate weight are selected for the active forging positions.

The delegate weight is defined as shown below.

```
delegate weight = minimal ( 10 * delegate self-vote , sum of all votes for the delegate )
```

Where `delegate self-vote` is the amount the delegate voted for its own account.

The `sum of all votes for the delegate` includes the self-vote.

Registering as delegate

In a DPoS system, each account that has an adequate enough balance to send a [register delegate transaction] can register a new delegate on the network. Other accounts can [vote for delegates] to support them with their stake. As a reward for forging, the delegate and its voters receive the [transaction fees and block rewards] of the forged blocks, and the transactions included in the forged blocks.

> [!info]
> The chosen delegate name has to be unique in the network.

Example: Sending a delegate registration transaction

```
{ type: 2, fee: '2000000000', recipientId: '', amount: '0', asset: { username: 'lightcurve' }, signatures: [] }
```

Voting for a delegate

Accounts can vote for delegates by sending a [vote delegate transaction]. The tokens used to vote for delegates will be locked. The locked tokens can be unlocked again if required, by [# unvoting the delegate again].

For instance, lets assume you hold an account with 1000 tokens.

It is possible to use only some of your tokens for voting, however, let's assume you want to vote with all 1000 tokens. In a scenario where you vote for 33 delegates, you would have to split your 1000 tokens, so that you can vote for each of them. For example, you could vote for 30 delegates with 31 tokens, or alternatively vote for ten delegates with the full amount of 100 tokens.

In the example shown below we decided to vote for one delegate with 73 tokens, and for another one with 30 tokens.

Example: Vote for two delegates

```
{ type: 3, fee: '2000000000', recipientId: '', amount: '0', asset: { votes: [ { delegateAddress: '17766529260000000L', amount: '7300000000' }, { delegateAddress: '6844721989000000L', amount: '3000000000' } ] }, signatures: [] }
```

① Votes 73 tokens and adds 73 tokens delegate weight to the delegate with address `17766529260000000L`.
② Votes 30 tokens and adds 30 tokens delegate weight to the delegate with address `6844721989000000L`.

Unvoting delegates and unlocking of tokens

The amount of tokens used for voting is locked and cannot be used for any other transactions. This includes but is not limited to further voting, balance transfers or transaction fees.

To use the locked tokens again, the account has to submit an [unvote delegate transaction], with a **negative amount** (also called [# negative vote]). Additionally, the account has to wait for the delay for unlocking 2000 blocks (approximately 5 hours and 30 minutes).

To remove the locked tokens, the account has to submit two transactions:

1. First, the tokens have to be unvoted. This is done with a new `VoteTransaction`, the transaction just needs to contain a negative amount. The tokens are now in an "unlocking" state. They have been unvoted but are not usable yet.

Example: Unvote a delegate

```
{ type: 3, fee: '2000000000', recipientId: '', amount: '0', asset: { votes: [ { delegateAddress: '6844721989000000L', amount: '-3000000000' } ] }, signatures: [] }
```

① Unvotes 30 tokens from the delegate with address `6844721989000000L`.

2. After a 2000 block period, the tokens can be unlocked. This is done with a new `UnlockTransaction`, the [input asset transaction] specifies which tokens have to be unlocked and added back to the balance. This mechanism is necessary because the tokens are not automatically unlocked because of the Lisk blockchain (particularly on the database level), could handle this in each transaction unnecessary.

Example: Unlock a delegate

```
{ type: 4, fee: '2000000000', recipientId: '', amount: '0', asset: { unlockObjects: [ { delegateAddress: '6844721989000000L', amount: '3000000000', unvoteHeight: '1234' } ] }, signatures: [] }
```

① Unlocks 30 tokens from the delegate with address `6844721989000000L`.

> [!info]
> **A token unlock transaction can contain multiple unlock objects**
>
> This allows an account to submit multiple delegate votes and receive those tokens with a single unlock transaction. Of course, all tokens must have been in the unlocking state for at least 2000 blocks for the unlock transaction to be valid.

BFT

The Byzantine Fault Tolerance (BFT) algorithm ensures that the network can reach consensus about the current state of the blockchain. This means that for a given height, eventually all Lisk nodes agree on the same block. This is particularly important in the case of forks, where different nodes have different views of the chain height, which can cause due to network latency or delegates forging multiple blocks in the same height.

Additionally, delegates can be punished by anyone in the network if they forge contradicting blocks, i.e., two blocks with consensus votes that violate the BFT protocol (see the [Lisk protocol documentation] for details). In order to avoid that a delegate is punished, a node operator has to take great care when enabling forging on a node. This means that it is very important to not run the same forging node twice.

How to discover misbehavior of a delegate

Any misbehavior of a delegate is indicated by two contradicting block headers signed by the delegate. A misbehavior can therefore be discovered by providing the two contradicting block headers. The node below checks if two contradicting block headers exist.

Detecting Contradicting Block Headers

```
// Check if the two block headers contradict each other.
if (
  header1.generatorPublicKey === header2.generatorPublicKey &&
  header1.blockSignature !== header2.blockSignature &&
  (header1.previousBlockId === header2.previousBlockId || header1.height === header2.height)
) {
  // Check if the signatures are valid.
  const isValidBlockSignature = validateBlockSignature(networkIdentifier, header1, header1.blockSignature, blockBytes);
  const isValidBlockSignature = validateBlockSignature(networkIdentifier, header2, header2.blockSignature, blockBytes);

  // If one of them is invalid, ignore it.
  if (!isValidBlockSignature || !isValidBlockSignature) {
    throw new Error('Block signature is not valid');
  }

  // Check if the delegate is actually in the active round.
  const activeDelegates = getActiveDelegates(header1.height);
  if (!activeDelegates.includes(header1.generatorPublicKey)) {
    throw new Error('Delegate is not in active round.');
  }

  // Check if the header1 is actually higher than header2.
  if (header1.height < header2.height) {
    throw new Error('Header1 is lower than header2.');
  }

  // Check if the header1 is actually higher than header2.
  if (header1.height === header2.height) {
    throw new Error('Headers have the same height.');
  }

  // Check if the header1 is actually higher than header2.
  if (header1.height > header2.height) {
    throw new Error('Header1 is higher than header2.');
  }

  // Check if the header1 is actually higher than header2.
  if (header1.height === header2.height) {
    throw new Error('Header1 is higher than header2
```

Plugins overview

An overview of all available plugins for the Lisk SDK.

Name	Included in the Lisk SDK by default?	Description
Dashboard plugin	no	Provides a user interface which enables developers to conveniently view and communicate with the blockchain application during the development.
Faucet plugin	no	Provides a user interface to receive a certain amount of tokens on request.
Forger plugin	yes	Monitors configured delegates forging activity, and voters information.
HTTP API plugin	yes	Provides basic HTTP API endpoints to get the running node information.
Monitor plugin	yes	Provides network statistics of the running node.
Report misbehavior plugin	yes	Provides automatic detection of delegate misbehavior and sends a reportDelegateMisbehaviorTransaction to the running node.

Dashboard plugin

`@liskhq/lisk-framework-dashboard-plugin` is a plugin for the Lisk Framework that provides a user interface which enables developers to conveniently view and communicate with the blockchain application during the development.

The dashboard plugin includes the following features:

- Monitor recent blocks and transactions
- Show created information about accounts
- Create new accounts
- Call exposed application actions
- Listen to exposed application events
- Create transactions
- Get node information

Alias	`dashboard`
Actions	none
Events	none

Table of Contents

Registering the plugin

Plugin configuration

- Available configuration options

Registering the plugin

How to register the dashboard plugin with the blockchain application.

```
const { Application, genesisBlockDevnet, configDevnet } = require('lisk-sdk');
const { DashboardPlugin } = require('@liskhq/lisk-framework-dashboard-plugin');

const app = Application.defaultApplication(genesisBlockDevnet, configDevnet);

app.registerPlugin(DashboardPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Plugin configuration

The dashboard plugin can be configured in the configuration of the blockchain application under `plugins.dashboard`.

> **Note**
>
> The application will use the default values in the case whereby there is no plugin configuration provided.

Dashboard options in the configuration

```
const { Application, genesisBlockDevnet, configDevnet } = require('lisk-sdk');
const { DashboardPlugin } = require('@liskhq/lisk-framework-dashboard-plugin');

const appConfig = utils.objects.mergeDeep({}, configDevnet, {
    plugins: {
        dashboard: {
            applicationUrl: 'ws://localhost:8080/ws',
            port: 4005,
            host: '127.0.0.1',
            applicationName: 'Lisk',
        }
    }
});

const app = Application.defaultApplication(genesisBlockDevnet, appConfig);

app.registerPlugin(DashboardPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Available configuration options

```
{
    applicationName: {
        type: 'string',
        description: 'Application name to be shown near Logo',
    },
    applicationUrl: {
        type: 'string',
        format: 'uri',
        description: 'URL to connect',
    },
    port: {
        type: 'integer',
        minimum: 1,
        maximum: 65535,
    },
    host: {
        type: 'string',
        format: 'ip',
    },
}
```

Faucet plugin

`@liskhq/lisk-framework-faucet-plugin` is a plugin for the Lisk Framework that provides a user interface to receive an certain amount of tokens on request. The faucet plugin is a useful tool during development and also in testing networks.

The faucet plugin includes the following features:

- enables sending tokens to different accounts from a faucet for testing
- allows customizing the faucet UI with a custom logo and application URL

Alias	`faucet`
Actions	⌄ faucet:authorize ⌄ faucet:fundTokens
Events	none

Table of Contents

Registering the plugin

Plugin configuration

- Available configuration options

Actions

- faucet:authorize
- faucet:fundTokens

Registering the plugin

How to register the faucet plugin with the blockchain application.

```
const { Application, genesisBlockDevnet, configDevnet } = require('lisk-sdk');
const { FaucetPlugin } = require('@liskhq/lisk-framework-faucet-plugin');

const app = Application.defaultApplication(genesisBlockDevnet, configDevnet);

app.registerPlugin(FaucetPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Plugin configuration

The faucet plugin can be configured in the configuration of the blockchain application under `plugins.faucet`.

Important

There are three required config options that need to be defined in order for the faucet plugin to function correctly:

1. `encryptedPassphrase`: The passphrase of the genesis account which is encrypted with encryptPassphraseWithPassword()
2. `captchaSecretkey`: The re-captcha secret key. You can register here to get valid keys for the recaptcha: https://developers.google.com/recaptcha/intro. For testing purposes it is not necessary to use your own keys, simply use the test API keys recaptcha.
3. `captchaSitekey`: The re-captcha site key.

Faucet options in the configuration

```
const { Application, genesisBlockDevnet, configDevnet } = require('lisk-sdk');
const { FaucetPlugin } = require('@liskhq/lisk-framework-faucet-plugin');

const appConfig = utils.objects.mergeDeep({}, configDevnet, {
    plugins: {
        faucet: {
            encryptedPassphrase: "",
            captchaSitekey: "",
            captchaSecretkey: "",
            port: 4004,
            host: '127.0.0.1',
            applicationUrl: 'ws://localhost:8080/ws',
            fee: '0.1',
            amount: '100',
            tokenPrefix: 'lsk',
        }
    }
});

const app = Application.defaultApplication(genesisBlockDevnet, appConfig);

app.registerPlugin(FaucetPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Available configuration options

```
{
    port: {
        type: 'integer',
        minimum: 1,
        maximum: 65535,
    },
    host: {
        type: 'string',
        format: 'ip',
    },
    encryptedPassphrase: {
        type: 'string',
        format: 'encryptedPassphrase',
        description: 'Encrypted passphrase of the genesis account',
    },
    applicationUrl: {
        type: 'string',
        format: 'uri',
        description: 'URL to connect',
    },
    fee: {
        type: 'string',
        description: 'The transaction fee used to faucet an account',
    },
    amount: {
        type: 'string',
        description: 'Number of tokens to fund an account per request',
    },
    tokenPrefix: {
        type: 'string',
        description: 'The token prefix associated with your application',
    },
    logoURL: {
        type: 'string',
        format: 'uri',
        description: 'The URL of the logo used on the UI',
    },
    captchaSecretkey: {
        type: 'string',
        description: 'The re-captcha secret key',
    },
    captchaSitekey: {
        type: 'string',
        description: 'The re-captcha site key',
    },
}
```

Actions

faucet:authorize

Enables and disables the faucet.

Example

```
const { apiClient } = require('lisk-sdk');

let clientCache;

const getClient = async () => {
  if (!clientCache) {
    clientCache = await apiClient.createWSClient('ws://localhost:8080/ws');
  }
  return clientCache;
};
const useClient = async () => {
  const client = await getClient();
  const input = {
    password: "secretPassword",
    enable: true
  };
  const authorize = await client.invoke('faucet:authorize', input);
  return authorize;
};
useClient().then((val) => {
  console.log("val:",val);
});
/*
{
  "result": "Successfully enabled the faucet."
}
 */
```

Input

```
{
    password: string,
    enable: boolean
}
```

Returns

```
{ result: string }
```

faucet:fundTokens

Example

```
const { apiClient } = require('lisk-sdk');

let clientCache;

const getClient = async () => {
  if (!clientCache) {
    clientCache = await apiClient.createWSClient('ws://localhost:8080/ws');
  }
  return clientCache;
};
const useClient = async () => {
  const client = await getClient();
  const input = {
    address: ""
  };
  const fund = await client.invoke('faucet:fundTokens', input);
  return fund;
};
useClient().then((val) => {
  console.log("val:",val);
});
```

Input

```
{
    address: string,
    token: string
}
```

Returns

```
{ result: string }
```

Forger plugin

`@liskhq/lisk-framework-forger-plugin` is a plugin for the Lisk Framework that monitors configured delegates forging activity, and voters information.

Alias	`forger`
Actions	• forger:getVoters • forger:getForgingInfo
Events	• forger:block:created • forger:block:missed

Table of Contents

Registering the plugin

Default configuration

Actions

- forger:getVoters
- forger:getForgingInfo

Events

- forger:block:created
- forger:block:missed

Registering the plugin

How to register the forger plugin with the blockchain application.

```
// Import the ForgerPlugin from the lisk-sdk package
const { Application, ForgerPlugin, genesisBlockDevnet, configDevnet } = require('lisk-sdk');

const app = Application.defaultApplication(genesisBlockDevnet, configDevnet);

// Register the plugin with the application
app.registerPlugin(ForgerPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Default configuration

The forger plugin can be configured in the configuration of the blockchain application under `plugins.forger`.

> **Note**
>
> The application will use the default values, in case no plugin configuration is provided.

Forger options in the configuration

```
const { Application, ForgerPlugin, genesisBlockDevnet, configDevnet } = require('lisk-sdk');

configDevnet.plugins.forger = {};
// Configure where to store the plugin data.
// Default location is ~/.lisk/forger
configDevnet.plugins.forger.dataPath = '~/.lisk/forger'; (1)

const app = Application.defaultApplication(genesisBlockDevnet, configDevnet);

app.registerPlugin(ForgerPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

① `dataPath`: The data path for storing forging related information captured from the application.

Actions

forger:getVoters

Returns the calculated voters for all forgers registered in the network.

Input

none

Returns

```
{
    "address": "a6f6a0543ae470c6b056021cb2ac153368eafeec",
    "username": "genesis_2",
    "totalVotesReceived": "1002000000000",
    "voters": [
      {
        "address": "a28d5e34007fd8fe6d7903044eb23a60fdad3c00",
        "amount": "2000000000"
      }
    ]
}[]
```

forger:getForgingInfo

Returns the status of all forgers registered in the network.

Input

none

Returns

```
{
  "address": "a28d5e34007fd8fe6d7903044eb23a60fdad3c00",
  "forging": true,
  "username": "genesis_1",
  "totalReceivedFees": "0",
  "totalReceivedRewards": "0",
  "totalProducedBlocks": 4,
  "totalVotesReceived": "1000000000000",
  "consecutiveMissedBlocks": 176
}[]
```

Events

forger:block:created

Returns forging information about the latest forged block.

Message

```
{
    reward,
    forgerAddress,
    height,
    timestamp: Date.now(),
}
```

forger:block:missed

Returns an object with information about how many blocks have been missed by all delegates up to the current height of the blockchain.

Message

```
{
    {
      address: missedBlocks, (1)
      address2: missedBlocks,
      // ...
    }
    height, (2)
    timestamp: date, (3)
}
```

① Number of missed blocks by delegate address.

② Current height of the blockchain.

③ Current time.

HTTP API plugins

Introduced in Kong Gateway 3.0 and enabled in Kong Konnect, a plugin is the Gen 3 resource that adds new HTTP API capabilities to Kong Gateway.

Key	Value
Type	Plugin
Family	API

Table of Contents

- Registering the plugin
- Global configuration
- API specification
  - Plugin
  - Request body schema
  - Response body schema
  - Validation
  - Examples
    - Plugin
    - Request body schema
    - Response body schema
    - Validation
- Entities
  - Plugin
    - Plugin: fields
    - Plugin: custom fields by protocol
    - Plugin: nested fields
    - Plugin: foreign relationships
    - Plugin: abstract attributes
    - Plugin: primary key
    - Plugin: cache key
    - Plugin: endpoint key
  - Consumer
    - Consumer: fields
    - Consumer: primary key
    - Consumer: cache key
    - Consumer: endpoint key
  - Consumer Group
    - Consumer Group: fields
    - Consumer Group: primary key
    - Consumer Group: cache key
    - Consumer Group: endpoint key
  - Consumer Group Consumer
    - Consumer Group Consumer: fields
    - Consumer Group Consumer: primary key
    - Consumer Group Consumer: cache key
    - Consumer Group Consumer: endpoint key
  - Consumer Group Plugin
    - Consumer Group Plugin: fields
    - Consumer Group Plugin: primary key
    - Consumer Group Plugin: cache key
    - Consumer Group Plugin: endpoint key

Registering the plugin

Add a plugin record with the plugin name and desired configuration:

```
curl -X POST http://localhost:8001/plugins \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --data "name=<PLUGIN_NAME>" \
  --data "config.<PARAMETER_NAME>=<PARAMETER_VALUE>"
```

```
curl -X POST http://localhost:8001/plugins \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --data "name=<PLUGIN_NAME>" \
  --data "config.<PARAMETER_NAME>=<PARAMETER_VALUE>"
```

Global configuration

Plugins can be configured to be enabled on every service in the Gateway cluster, or only [**scoped to Routes**](#) or Services.

Configuring a plugin on a Service means that it will be executed on all the Routes of that Service. Configuring a plugin on a Route means that the plugin will only be executed when that Route is matched. If a plugin is configured on both a Service and a Route, both configurations will be taken into account.

```
curl -X POST http://localhost:8001/services/<SERVICE_NAME_OR_ID>/plugins \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --data "name=<PLUGIN_NAME>" \
  --data "config.<PARAMETER_NAME>=<PARAMETER_VALUE>"
```

```
curl -X POST http://localhost:8001/services/<SERVICE_NAME_OR_ID>/plugins \
  --header "Content-Type: application/x-www-form-urlencoded" \
  --data "name=<PLUGIN_NAME>" \
  --data "config.<PARAMETER_NAME>=<PARAMETER_VALUE>"
```

Configuring a plugin on a Consumer means that it will be executed on requests made by that Consumer, often identified by the authentication credentials being used. Configuring a plugin on a Consumer Group means that it will be executed on requests made by any Consumer belonging to that Consumer Group.

Finally, configuring a plugin on a Consumer Group Consumer means that it will be executed on requests made by that Consumer when it belongs to that Consumer Group.

Plugins can be configured in many contexts, but only one will run. Kong Gateway will run the most specific configuration available. The specificity rules are, in order:

1. **Consumer Group Consumer**: Plugins configured on a Consumer Group Consumer will run before all other configurations.
2. **Consumer**: Plugins configured on a Consumer will run if no Consumer Group Consumer configuration is available.
3. **Consumer Group**: Plugins configured on a Consumer Group will run if no Consumer or Consumer Group Consumer configuration is available.
4. **Route and Consumer**: Plugins configured on a Route and a Consumer will run if no Consumer Group or Consumer configuration is available.
5. **Service and Consumer**: Plugins configured on a Service and a Consumer will run if no Route, Consumer Group, or Consumer configuration is available.
6. **Route and Consumer Group**: Plugins configured on a Route and a Consumer Group will run if no Consumer configuration is available.
7. **Service and Consumer Group**: Plugins configured on a Service and a Consumer Group will run if no Route or Consumer configuration is available.
8. **Route**: Plugins configured on a Route will run if no Consumer or Consumer Group configuration is available.
9. **Service**: Plugins configured on a Service will run if no Route, Consumer, or Consumer Group configuration is available.
10. **Consumer Group**: Plugins configured on a Consumer Group will run if no Consumer configuration is available.
11. **Global**: Plugins configured globally will run if no Route, Service, Consumer, or Consumer Group configuration is available.

Plugins can be configured to be enabled on every service in the Gateway cluster, or only [**scoped to Routes**](#) or Services.

Configuring a plugin on a Service means that it will be executed on all the Routes of that Service. Configuring a plugin on a Route means that the plugin will only be executed when that Route is matched. If a plugin is configured on both a Service and a Route, both configurations will be taken into account.

Configuring a plugin on a Consumer means that it will be executed on requests made by that Consumer, often identified by the authentication credentials being used. Configuring a plugin on a Consumer Group means that it will be executed on requests made by any Consumer belonging to that Consumer Group.

Finally, configuring a plugin on a Consumer Group Consumer means that it will be executed on requests made by that Consumer when it belongs to that Consumer Group.

Plugins can be configured in many contexts, but only one will run. Kong Gateway will run the most specific configuration available. The specificity rules are, in order:

1. **Consumer Group Consumer**: Plugins configured on a Consumer Group Consumer will run before all other configurations.
2. **Consumer**: Plugins configured on a Consumer will run if no Consumer Group Consumer configuration is available.
3. **Consumer Group**: Plugins configured on a Consumer Group will run if no Consumer or Consumer Group Consumer configuration is available.
4. **Route and Consumer**: Plugins configured on a Route and a Consumer will run if no Consumer Group or Consumer configuration is available.
5. **Service and Consumer**: Plugins configured on a Service and a Consumer will run if no Route, Consumer Group, or Consumer configuration is available.
6. **Route and Consumer Group**: Plugins configured on a Route and a Consumer Group will run if no Consumer configuration is available.
7. **Service and Consumer Group**: Plugins configured on a Service and a Consumer Group will run if no Route or Consumer configuration is available.
8. **Route**: Plugins configured on a Route will run if no Consumer or Consumer Group configuration is available.
9. **Service**: Plugins configured on a Service will run if no Route, Consumer, or Consumer Group configuration is available.
10. **Consumer Group**: Plugins configured on a Consumer Group will run if no Consumer configuration is available.
11. **Global**: Plugins configured globally will run if no Route, Service, Consumer, or Consumer Group configuration is available.

API specification

Plugin

A plugin entity is defined by its name, its configuration, and optionally by an association to a Route, a Service, a Consumer, a Consumer Group, or a Consumer Group Consumer. This entity is used to configure Kong to perform certain actions during the request/response lifecycle.

Creating two plugin entities with the same name and the same set of associations is invalid.

Request body schema

Request body schema for [**application/json**](#). The request body can be submitted in multiple shapes:

```
{
  "name": "...",
  "instance_name": "...",
  "service": {
    "id": "..."
  },
  "route": {
    "id": "..."
  },
  "consumer": {
    "id": "..."
  },
  "consumer_group": {
    "id": "..."
  },
  "consumer_group_consumer": {
    "id": "..."
  },
  "config": {
    "...": "..."
  },
  "protocols": [
    "..."
  ],
  "enabled": true,
  "tags": [
    "..."
  ],
  "ordering": {
    "before": {
      "access": [
        "..."
      ]
    },
    "after": {
      "access": [
        "..."
      ]
    }
  }
}
```

Response body schema

Response body schema for [**application/json**](#).

Validation

Validation rules for the request body:

Type	Key	Description	Value(s)
Required	**name**	The name of the plugin to apply.	string
Required	**config**	The configuration properties for the plugin. Each plugin has its own set of configuration properties. Refer to the documentation for each plugin to know which properties to set.	object
Mutually exclusive	**service**	The Service this plugin is associated with. This Service will be the context for the execution of this plugin.	object
Mutually exclusive	**route**	The Route this plugin is associated with. This Route will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer**	The Consumer this plugin is associated with. This Consumer will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer_group**	The Consumer Group this plugin is associated with. This Consumer Group will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer_group_consumer**	The Consumer Group Consumer this plugin is associated with. This Consumer Group Consumer will be the context for the execution of this plugin.	object
Optional	**instance_name**	An optional custom name to identify an instance of the plugin.	string
Optional	**protocols**	A list of the request protocols that will trigger this plugin. The default value, depending on the plugin type, is usually all supported protocols.	array of strings
Optional	**enabled**	Whether the plugin is applied.	boolean Default: true
Optional	**tags**	An optional set of strings associated with the plugin for grouping and filtering.	array of strings
Optional	**ordering**	Controls the order in which the plugin is executed relative to other plugins.	object

Examples

Plugin

Request body schema for [**application/json**](#).

Request body:

```
{
  "name": "rate-limiting",
  "config": {
    "minute": 5,
    "policy": "local"
  }
}
```

Response body:

```
{
  "id": "3473c251-5b6c-4f45-b1ff-7ede735a366d",
  "name": "rate-limiting",
  "instance_name": null,
  "route": null,
  "service": null,
  "consumer": null,
  "consumer_group": null,
  "config": {
    "minute": 5,
    "policy": "local"
  },
  "protocols": ["grpc", "grpcs", "http", "https"],
  "enabled": true,
  "tags": null,
  "ordering": null,
  "created_at": 1422386534
}
```

Request body schema

Request body schema for [**application/json**](#).

Request body:

```
{
  "name": "rate-limiting",
  "config": {
    "minute": 5,
    "policy": "local"
  }
}
```

Response body:

```
{
  "id": "3473c251-5b6c-4f45-b1ff-7ede735a366d",
  "name": "rate-limiting",
  "instance_name": null,
  "route": null,
  "service": null,
  "consumer": null,
  "consumer_group": null,
  "config": {
    "minute": 5,
    "policy": "local"
  },
  "protocols": ["grpc", "grpcs", "http", "https"],
  "enabled": true,
  "tags": null,
  "ordering": null,
  "created_at": 1422386534
}
```

Response body schema

Response body schema for [**application/json**](#).

Response body:

```
{
  "id": "3473c251-5b6c-4f45-b1ff-7ede735a366d",
  "name": "rate-limiting",
  "instance_name": null,
  "route": null,
  "service": null,
  "consumer": null,
  "consumer_group": null,
  "config": {
    "minute": 5,
    "policy": "local"
  },
  "protocols": ["grpc", "grpcs", "http", "https"],
  "enabled": true,
  "tags": null,
  "ordering": null,
  "created_at": 1422386534
}
```

Validation

Validation rules for the request body:

Type	Key	Description	Value(s)
Required	**name**	The name of the plugin to apply.	string
Required	**config**	The configuration properties for the plugin. Each plugin has its own set of configuration properties. Refer to the documentation for each plugin to know which properties to set.	object
Mutually exclusive	**service**	The Service this plugin is associated with. This Service will be the context for the execution of this plugin.	object
Mutually exclusive	**route**	The Route this plugin is associated with. This Route will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer**	The Consumer this plugin is associated with. This Consumer will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer_group**	The Consumer Group this plugin is associated with. This Consumer Group will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer_group_consumer**	The Consumer Group Consumer this plugin is associated with. This Consumer Group Consumer will be the context for the execution of this plugin.	object
Optional	**instance_name**	An optional custom name to identify an instance of the plugin.	string
Optional	**protocols**	A list of the request protocols that will trigger this plugin. The default value, depending on the plugin type, is usually all supported protocols.	array of strings
Optional	**enabled**	Whether the plugin is applied.	boolean Default: true
Optional	**tags**	An optional set of strings associated with the plugin for grouping and filtering.	array of strings
Optional	**ordering**	Controls the order in which the plugin is executed relative to other plugins.	object

Examples

Plugin

Request body schema for [**application/json**](#).

Request body:

```
{
  "name": "rate-limiting",
  "config": {
    "minute": 5,
    "policy": "local"
  }
}
```

Response body:

```
{
  "id": "3473c251-5b6c-4f45-b1ff-7ede735a366d",
  "name": "rate-limiting",
  "instance_name": null,
  "route": null,
  "service": null,
  "consumer": null,
  "consumer_group": null,
  "config": {
    "minute": 5,
    "policy": "local"
  },
  "protocols": ["grpc", "grpcs", "http", "https"],
  "enabled": true,
  "tags": null,
  "ordering": null,
  "created_at": 1422386534
}
```

Request body schema

Request body schema for [**application/json**](#).

Request body:

```
{
  "name": "rate-limiting",
  "config": {
    "minute": 5,
    "policy": "local"
  }
}
```

Response body:

```
{
  "id": "3473c251-5b6c-4f45-b1ff-7ede735a366d",
  "name": "rate-limiting",
  "instance_name": null,
  "route": null,
  "service": null,
  "consumer": null,
  "consumer_group": null,
  "config": {
    "minute": 5,
    "policy": "local"
  },
  "protocols": ["grpc", "grpcs", "http", "https"],
  "enabled": true,
  "tags": null,
  "ordering": null,
  "created_at": 1422386534
}
```

Response body schema

Response body schema for [**application/json**](#).

Response body:

```
{
  "id": "3473c251-5b6c-4f45-b1ff-7ede735a366d",
  "name": "rate-limiting",
  "instance_name": null,
  "route": null,
  "service": null,
  "consumer": null,
  "consumer_group": null,
  "config": {
    "minute": 5,
    "policy": "local"
  },
  "protocols": ["grpc", "grpcs", "http", "https"],
  "enabled": true,
  "tags": null,
  "ordering": null,
  "created_at": 1422386534
}
```

Validation

Validation rules for the request body:

Type	Key	Description	Value(s)
Required	**name**	The name of the plugin to apply.	string
Required	**config**	The configuration properties for the plugin. Each plugin has its own set of configuration properties. Refer to the documentation for each plugin to know which properties to set.	object
Mutually exclusive	**service**	The Service this plugin is associated with. This Service will be the context for the execution of this plugin.	object
Mutually exclusive	**route**	The Route this plugin is associated with. This Route will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer**	The Consumer this plugin is associated with. This Consumer will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer_group**	The Consumer Group this plugin is associated with. This Consumer Group will be the context for the execution of this plugin.	object
Mutually exclusive	**consumer_group_consumer**	The Consumer Group Consumer this plugin is associated with. This Consumer Group Consumer will be the context for the execution of this plugin.	object
Optional	**instance_name**	An optional custom name to identify an instance of the plugin.	string
Optional	**protocols**	A list of the request protocols that will trigger this plugin. The default value, depending on the plugin type, is usually all supported protocols.	array of strings
Optional	**enabled**	Whether the plugin is applied.	boolean Default: true
Optional	**tags**	An optional set of strings associated with the plugin for grouping and filtering.	array of strings
Optional	**ordering**	Controls the order in which the plugin is executed relative to other plugins.	object

Entities

Plugin

Plugin: fields

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid
Required	**name**	The name of the Plugin that is going to be applied, like `key-auth` or `basic-auth`. The Plugin must be installed and loaded by Kong Gateway in advance.	string
Optional	**instance_name**	An optional custom name to identify an instance of the plugin.	string
Optional	**route**	If set, the plugin will only activate when this Route receives a request. Specify a value of type `route` with the primary key of the Route to associate the plugin with.	record Route
Optional	**service**	If set, the plugin will only activate when this Service receives a request. Specify a value of type `service` with the primary key of the Service to associate the plugin with.	record Service
Optional	**consumer**	If set, the plugin will only activate when this Consumer receives a request. Specify a value of type `consumer` with the primary key of the Consumer to associate the plugin with.	record Consumer
Optional	**consumer_group**	If set, the plugin will only activate when this Consumer Group receives a request. Specify a value of type `consumer_group` with the primary key of the Consumer Group to associate the plugin with.	record Consumer Group
Optional	**config**	The configuration properties for the Plugin. Each plugin has its own set of configuration properties. Refer to the documentation for each plugin to know which properties to set.	record Plugin Config
Optional	**protocols**	A list of the request protocols that will trigger this plugin. The default value, depending on the plugin type, is usually all supported protocols.	array of strings enum "grpc", "grpcs", "http", "https", "tcp", "tls", "tls_passthrough", "udp", "ws", "wss"
Optional	**enabled**	Whether the plugin is applied.	boolean Default: true
Optional	**tags**	An optional set of strings associated with the plugin for grouping and filtering.	array of strings
Optional	**ordering**	Controls the order in which the plugin is executed relative to other plugins.	object
Optional	**created_at**	Unix epoch when the resource was created.	integer
Optional	**updated_at**	Unix epoch when the resource was last updated.	integer

Plugin: custom fields by protocol

Type	Key	Description	Value(s)
Optional	**ws_id**	The ID of the workspace the plugin is in.	string uuid
Optional	**cache_key**	A string representing a plugin and its associated entities, to be used for caching purposes.	string

Plugin: nested fields

Type	Key	Description	Value(s)
Optional	**ordering.before**	Controls the order in which the plugin is executed relative to other plugins.	object
Optional	**ordering.before.access**	Controls the order in which the plugin is executed relative to other plugins.	array of strings
Optional	**ordering.after**	Controls the order in which the plugin is executed relative to other plugins.	object
Optional	**ordering.after.access**	Controls the order in which the plugin is executed relative to other plugins.	array of strings

Plugin: foreign relationships

Type	Key	Description	Value(s)
Optional	**route.id**	The UUID of the Route.	string uuid
Optional	**service.id**	The UUID of the Service.	string uuid
Optional	**consumer.id**	The UUID of the Consumer.	string uuid
Optional	**consumer_group.id**	The UUID of the Consumer Group.	string uuid

Plugin: abstract attributes

Type	Key	Description	Value(s)
Optional	**route.***	Nested attributes of the Route this Plugin is applied to.	
Optional	**service.***	Nested attributes of the Service this Plugin is applied to.	
Optional	**consumer.***	Nested attributes of the Consumer this Plugin is applied to.	
Optional	**consumer_group.***	Nested attributes of the Consumer Group this Plugin is applied to.	

Plugin: primary key

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid

Plugin: cache key

Type	Key	Description	Value(s)
Unique	**cache_key**	A string representing a plugin and its associated entities, to be used for caching purposes.	string

Plugin: endpoint key

Type	Key	Description	Value(s)
Unique	**name**	The name of the Plugin that is going to be applied, like `key-auth` or `basic-auth`. The Plugin must be installed and loaded by Kong Gateway in advance.	string
Unique	**instance_name**	An optional custom name to identify an instance of the plugin.	string

Consumer

Consumer: fields

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid
Optional	**username**	The unique username of the Consumer. You must send either this field or `custom_id` with the request.	string
Optional	**custom_id**	Stores the existing unique ID of the Consumer. You must send either this field or `username` with the request.	string
Optional	**tags**	An optional set of strings associated with the Consumer for grouping and filtering.	array of strings
Optional	**created_at**	Unix epoch when the resource was created.	integer
Optional	**updated_at**	Unix epoch when the resource was last updated.	integer

Consumer: primary key

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid

Consumer: cache key

Type	Key	Description	Value(s)
Unique	**id**	A string representing a UUID (universally unique identifier).	string uuid
Unique	**username**	The unique username of the Consumer. You must send either this field or `custom_id` with the request.	string
Unique	**custom_id**	Stores the existing unique ID of the Consumer. You must send either this field or `username` with the request.	string

Consumer: endpoint key

Type	Key	Description	Value(s)
Unique	**id**	A string representing a UUID (universally unique identifier).	string uuid
Unique	**username**	The unique username of the Consumer. You must send either this field or `custom_id` with the request.	string
Unique	**custom_id**	Stores the existing unique ID of the Consumer. You must send either this field or `username` with the request.	string

Consumer Group

Consumer Group: fields

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid
Required	**name**	The unique name of the Consumer Group.	string
Optional	**tags**	An optional set of strings associated with the Consumer Group for grouping and filtering.	array of strings
Optional	**created_at**	Unix epoch when the resource was created.	integer
Optional	**updated_at**	Unix epoch when the resource was last updated.	integer

Consumer Group: primary key

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid

Consumer Group: cache key

Type	Key	Description	Value(s)
Unique	**id**	A string representing a UUID (universally unique identifier).	string uuid
Unique	**name**	The unique name of the Consumer Group.	string

Consumer Group: endpoint key

Type	Key	Description	Value(s)
Unique	**id**	A string representing a UUID (universally unique identifier).	string uuid
Unique	**name**	The unique name of the Consumer Group.	string

Consumer Group Consumer

Consumer Group Consumer: fields

Type	Key	Description	Value(s)
Primary key	**consumer_group.id**	The UUID of the Consumer Group.	string uuid
Primary key	**consumer.id**	The UUID of the Consumer.	string uuid
Optional	**created_at**	Unix epoch when the resource was created.	integer
Optional	**updated_at**	Unix epoch when the resource was last updated.	integer

Consumer Group Consumer: primary key

Type	Key	Description	Value(s)
Primary key	**consumer_group.id**	The UUID of the Consumer Group.	string uuid
Primary key	**consumer.id**	The UUID of the Consumer.	string uuid

Consumer Group Consumer: cache key

Type	Key	Description	Value(s)
Unique	**consumer_group.id**	The UUID of the Consumer Group.	string uuid
Unique	**consumer.id**	The UUID of the Consumer.	string uuid

Consumer Group Consumer: endpoint key

Type	Key	Description	Value(s)
Unique	**consumer_group.id**	The UUID of the Consumer Group.	string uuid
Unique	**consumer.id**	The UUID of the Consumer.	string uuid

Consumer Group Plugin

Consumer Group Plugin: fields

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid
Required	**name**	The name of the Plugin that is going to be applied, like `key-auth` or `basic-auth`. The Plugin must be installed and loaded by Kong Gateway in advance.	string
Optional	**instance_name**	An optional custom name to identify an instance of the plugin.	string
Optional	**consumer_group**	If set, the plugin will only activate when this Consumer Group receives a request. Specify a value of type `consumer_group` with the primary key of the Consumer Group to associate the plugin with.	record Consumer Group
Optional	**config**	The configuration properties for the Plugin. Each plugin has its own set of configuration properties. Refer to the documentation for each plugin to know which properties to set.	record Plugin Config
Optional	**enabled**	Whether the plugin is applied.	boolean Default: true
Optional	**tags**	An optional set of strings associated with the plugin for grouping and filtering.	array of strings
Optional	**created_at**	Unix epoch when the resource was created.	integer
Optional	**updated_at**	Unix epoch when the resource was last updated.	integer

Consumer Group Plugin: primary key

Type	Key	Description	Value(s)
Primary key	**id**	A string representing a UUID (universally unique identifier).	string uuid

Consumer Group Plugin: cache key

Type	Key	Description	Value(s)
Unique	**id**	A string representing a UUID (universally unique identifier).	string uuid
Unique	**name**	The name of the Plugin that is going to be applied, like `key-auth` or `basic-auth`. The Plugin must be installed and loaded by Kong Gateway in advance.	string
Unique	**instance_name**	An optional custom name to identify an instance of the plugin.	string

Consumer Group Plugin: endpoint key

Type	Key	Description	Value(s)
Unique	**id**	A string representing a UUID (universally unique identifier).	string uuid
Unique	**name**	The name of the Plugin that is going to be applied, like `key-auth` or `basic-auth`. The Plugin must be installed and loaded by Kong Gateway in advance.	string
Unique	**instance_name**	An optional custom name to identify an instance of the plugin.	string

Monitor plugin

`@liskhq/lisk-framework-monitor-plugin` is a plugin for the Lisk Framework that provides network statistics of the running node.

Tip

The monitor plugin offers Prometheus endpoints, which are reachable under the endpoint `/api/prometheus/metrics`.

Alias	`monitor`
Actions	• monitor:getBlockStats • monitor:getTransactionStats • monitor:getNetworkStats • monitor:getForkStats
Events	none

Table of Contents

Registering the plugin

Default configuration

Actions

- monitor:getBlockStats
- monitor:getForkStats
- monitor:getNetworkStats
- monitor:getTransactionStats

Registering the plugin

How to register the monitor plugin with the blockchain application.

```
const { Application, MonitorPlugin, genesisBlockDevnet, configDevnet } = require('lisk-sdk');

const app = Application.defaultApplication(genesisBlockDevnet, configDevnet);

app.registerPlugin(MonitorPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Default configuration

The monitor plugin can be configured in the configuration of the blockchain application under `plugins.monitor`.

Note

The application will use the default values, in case there is no plugin configuration provided.

Monitor options in the configuration

```
const { Application, HTTPAPIPlugin, genesisBlockDevnet, configDevnet } = require('lisk-sdk');

const appConfig = utils.objects.mergeDeep({}, configDevnet, {
  plugins: {
    monitor: { (1)
                port: 4003, (2)
                whiteList: ['127.0.0.1'], (3)
                cors: { (4)
                        origin: '*', (5)
                        methods: ['GET', 'POST', 'PUT'], (6)
                },
                limits: { (7)
                        max: 0, (8)
                        delayMs: 0, (9)
                        delayAfter: 0, (10)
                        windowMs: 60000, (11)
                        headersTimeout: 5000, (12)
                        serverSetTimeout: 20000, (13)
                },
        },
  }
});

const app = Application.defaultApplication(genesisBlockDevnet, appConfig);

app.registerPlugin(HTTPAPIPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

① `monitor`: Contains options for the monitor plugin.

② `port`: HTTP port which the monitor plugin listens on.

③ `whiteList`: This parameter allows connections to the monitor plugin by IP.

④ `cors`: Contain options for CORS, (Cross-Origin Resource Sharing).

⑤ `origin`: Defines the domains that the resource can be accessed by in a cross-site manner. Defaults to all domains.

⑥ `methods` (optional): Defines the allowed methods for CORS.

⑦ `limits`: Contains options about time limits.

⑧ `max`: Maximum amount of connections.

⑨ `delayMs`: Minimum delay between API calls in ms.

⑩ `delayAfter`: Minimum delay after an API call in ms.

⑪ `windowMs`: Minimum delay between API calls from the same window.

⑫ `headersTimeout`: Indicates the minimum amount of time an idle connection has to be kept opened, (in seconds).

⑬ `serverSetTimeout`: Time to wait for a response from the server before timing out.

Actions

monitor:getBlockStats

Returns statistics about the blocks that occurred in the network so far.

Input

none

Returns

```
{
  "blocks": {},
  "averageReceivedBlocks": 0,
  "connectedPeers": 0
}
```

monitor:getForkStats

Returns statistics about the forks that occurred in the network so far.

Input

none

Returns

```
{
  "forkEventCount": 0,
  "blockHeaders": {}
}
```

monitor:getNetworkStats

Returns various statistics about the network the node is connected to.

Input

none

Returns

```
{
  "startTime": 1610646123535,
  "incoming": {
    "count": 0,
    "connects": 0,
    "disconnects": 0
  },
  "outgoing": {
    "count": 0,
    "connects": 0,
    "disconnects": 3
  },
  "banning": {
    "bannedPeers": {},
    "totalBannedPeers": 0
  },
  "totalErrors": 6,
  "totalPeersDiscovered": 0,
  "totalRemovedPeers": 3,
  "totalMessagesReceived": {},
  "totalRequestsReceived": {},
  "majorityHeight": {
    "height": 0,
    "count": 0
  },
  "totalPeers": {
    "connected": 0,
    "disconnected": 0
  }
}
```

monitor:getTransactionStats

Returns various statistics about transactions in the network.

Input

none

Returns

```
{
  "transactions": {},
  "connectedPeers": 0,
  "averageReceivedTransactions": 0
}
```

Report misbehavior plugin

`@liskhq/lisk-framework-report-misbehavior-plugin` is a plugin for the Lisk Framework that provides automatic detection of delegate misbehavior and sends a `reportDelegateMisbehaviorTransaction` to the running node.

Alias	`reportMisbehavior`
Actions	• reportMisbehavior:authorize
Events	none

Table of Contents

Registering the plugin

Default configuration

Actions

- reportMisbehavior:authorize

Registering the plugin

How to register the report misbehavior plugin with the blockchain application.

```
const { Application, ReportMisbehaviorPlugin, genesisBlockDevnet, configDevnet } = require('lisk-sdk');

const app = Application.defaultApplication(genesisBlockDevnet, configDevnet);

app.registerPlugin(ReportMisbehaviorPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

Default configuration

The report misbehavior plugin can be configured in the configuration of the blockchain application under `plugins.reportMisbehavior`.

> **Note**
>
> The application will use the default values, in case there is no plugin configuration provided.

Misbehavior reporting options in the configuration

```
const { Application, ReportMisbehaviorPlugin, genesisBlockDevnet, configDevnet } = require('lisk-sdk');

const appConfig = utils.objects.mergeDeep({}, configDevnet, {
  plugins: {
    reportMisbehavior: {
        dataPath: '~/.lisk/report-misbehavior', (1)
                cleanupFrequency: 3600, (2)
                encryptedPassphrase: '', (3)
                fee: 100000000 (4)
        },
  }
});

const app = Application.defaultApplication(genesisBlockDevnet, appConfig);

app.registerPlugin(ReportMisbehaviorPlugin);

app
  .run()
  .then(() => app.logger.info('App started...'))
  .catch(error => {
    console.error('Faced error in application', error);
    process.exit(1);
  });
```

① `dataPath`: The data path for storing misbehavior related information captured from the application.

② `cleanupFrequency` (optional): Frequency of cleaning up the outdated data in seconds.

③ `encryptedPassphrase` (optional): The encrypted passphrase of the account that will report the misbehavior.

④ `fee` (optional): The fee required to send a report misbehavior transaction.

Actions

reportMisbehavior:authorize

Invoke this action to enable or disable the delegate misbehavior report.

Input

```
{
  password: string,
  enable: boolean
}
```

Returns

```
"Successfully enabled/disabled the reporting of misbehavior."
```

Framework class interfaces

Table of Contents

Application

The BaseModule

The BaseAsset

The BasePlugin

Application

Interface of the Application class

```
interface Application {
    registerModule: (moduleClass: typeof BaseModule) => void; (1)
    registerPlugin: (pluginClass: typeof BasePlugin) => void; (2)
    run: () => Promise<void>(3)
    shutdown: (errorCode: number, message: string) => Promise<void> (4)
    networkIdentifier: Buffer; (5)
    getSchema: () => { (6)
        account: Schema;
        Block: Schema;
        blockHeader: Schema;
        blockHeaderAssets: { [blockVersion: number]: Schema };
        transaction: Schema;
        transactionsAssets: {
            moduleID: number;
            moduleName: string;
            assetID: number;
            assetName: string;
            schema: Schema;
        }
    }
}
```

① Adds a module to the application.

② Adds a plugin to the application.

③ Starts the application.

④ Shutdown stops the application with a given code and message.

⑤ NetworkIdentifer used to sign block and transactions.

⑥ getSchema returns all schema definitions that are registered to the application. Schema is a type defined by codec, which is a Lisk JSON schema.

The BaseModule

Interface of the BaseModule

```
import {
        GenesisConfig,
        AccountSchema,
        TransactionApplyContext,
        AfterBlockApplyContext,
        BeforeBlockApplyContext,
        AfterGenesisBlockApplyContext,
        Reducers,
        Actions,
        BaseModuleDataAccess,
} from '../types';
import { BaseAsset } from './base_asset';

interface Channel {
        publish(name: string, data?: object): void;
}

export abstract class BaseModule {
        public abstract name: string; (1)
        public abstract id: number; (2)
        public transactionAssets: BaseAsset[] = []; (3)
        public accountSchema?: AccountSchema; (4)
        public actions: Actions = {}; (5)
        public events: string[] = []; (6)
        public reducers: Reducers = {}; (7)
        public readonly config: GenesisConfig;
        protected _channel!: Channel;
        protected _dataAccess!: BaseModuleDataAccess;

        public constructor(genesisConfig: GenesisConfig) {
                this.config = genesisConfig;
        }

        public init(input: { channel: Channel; dataAccess: BaseModuleDataAccess }): void {
                this._channel = input.channel;
                this._dataAccess = input.dataAccess;
        }

        public async beforeTransactionApply?(context: TransactionApplyContext): Promise<void>(8)
        public async afterTransactionApply?(context: TransactionApplyContext): Promise<void>(9)
        public async afterGenesisBlockApply?(context: AfterGenesisBlockApplyContext): Promise<void>(10)
        public async beforeBlockApply?(context: BeforeBlockApplyContext): Promise<void>(11)
        public async afterBlockApply?(context: AfterBlockApplyContext): Promise<void>(12)
}
```

① Required. Name of the module.

② Required. Module ID.

③ Optional. List of transaction assets for this module.

④ Optional. Schema of account asset to be used in this module.

⑤ Optional. A set of actions which can be called from plugins and external services.

⑥ Optional. A set of events name that will be emitted from this module, which can be subscribed by plugins and external sevices.

⑦ Optional. A set of actions which can be called from other modules during the block processing.

⑧ Optional. A function which will be called for all transactions before the `apply` function of the transaction.

⑨ Optional. A function which will be called for all transactions after the `apply` function of the transaction.

⑩ Optional. A function which will be called while applying the genesis block.

⑪ Optional. A function which will be called before a block is applied.

⑫ Optional. A function which will be called after a block is applied.

The BaseAsset

Interface of the BaseAsset

```
class BaseAsset<K> {
  public name: string; (1)
  public id: number; (2)
  public schema: Schema; (3)
  public apply: ({senderID:Buffer, asset: K, stateStore: StateStore, transaction: Transaction }) => void; (
  public validate?: (asset: K) => void; (5)
}
```

① Name of the asset which will be used in the UI.

② Type of the asset which will be used in the transaction.

③ Schema of the content of this asset.

④ Optional. Apply.

⑤ Optional. Validation function that can be implemented if additional.

The BasePlugin

Interface of the BasePlugin

```
class BasePlugin {
    static get alias(): string; (1)
    static get info(): { (2)
      author: string,
      version: string,
      name: string,
    };
    get defaults(): Record<string, unknown>(3)
    get events(): string[]; (4)
    get actions(): { [actionName: string]: actionHandler }; (5)
    async load(channel): Promise<void>(6)
    async unload(): Promise<void>(7)
};
```

① A unique plugin identifier, that can be accessed throughout the system. If any plugins are already registered with the same alias, then an error will be thrown.

② Package meta information.

③ Supported configurations for the plugin with default values.

④ List of valid events which this plugin wants to register with the controller. Each event name will be prefixed by the plugin alias, e.g. `pluginName:event1`. Listing an event means to register the event in the application. Any plugin can subscribe or publish that event in the application.

⑤ Object of valid actions which this plugin wants to register with the controller. Each action name will be prefixed by the plugin alias, e.g. `pluginName:action1`. The source plugin can define the action while others can invoke that action.

⑥ Method which will be invoked by the controller to load the plugin. Ensure all loading logic gets completed during the life cycle of load. The controller emits an event `app:ready` which you can use to perform certain activities, which you may wish to perform when all other plugins are loaded.

⑦ Method to be invoked by the controller to perform the cleanup.

The Lisk SDK testing utilities

A detailed reference of the available test utilities in the Lisk SDK test suite.

Table of Contents

Fixtures
- defaultPassphraseAccount
- defaultAccounts
- createDefaultAccount
- defaultPassword
- defaultConfig
- getPassphraseFromDefaultConfig
- getValidatorFromDefaultConfig

Mocks
- StateStoreMock
- createMock
- createMock
- createMock
- createMock
- createMock

Functions
- getBlockProcessingEnv
- createDefaultApplicationEnv
- createTransaction
- createGenesisBlock
- getModuleInstance
- getAccountSchemaFromModules
- createBlock
- encodeBlockHeader
- createBlockHeaderWithDefaults
- createFakeBlockHeader
- createNewGenesisBlockApplyContext
- createBeforeBlockApplyContext
- createNewBlockApplyContext
- createTransactionApplyContext
- createApplyAssetContext
- createValidateAssetContext

Fixtures

defaultPassphraseAccount

Returns a default Account.

Usage

```
const { defaultPassphraseAccount } = require('lisk-framework/src/testing');
const defaultAccount = defaultPassphraseAccount();
```

- Returns

defaultAccounts

Returns a list of all addresses of the 101 genesis delegates.

Usage

```
const { defaultAccounts } = require('lisk-framework/src/testing');
const accounts = defaultAccounts();
```

- Returns

createDefaultAccount

Generates account data for a user account.

Parameters:

1. modulesArg (opt.) List of modules. The modules account schemas will be included in the returned account.
2. assetArgument Custom account data.

Usage

```
const { createDefaultAccount } = require('lisk-framework/src/testing');
const account = createDefaultAccount([ new TokenModule(), new AnotherModule() ], { token: { balance: BigInt(1000000000) } });
```

- Returns

defaultPassword

Returns a default password.

Usage

```
const { defaultPassword } = require('lisk-framework/src/testing');
const password = defaultPassword();
```

- Returns

defaultConfig

A default configuration for a blockchain application.

Usage

```
const { defaultConfig } = require('lisk-framework/src/testing');
const config = defaultConfig();
```

- Returns

getPassphraseFromDefaultConfig

Returns the passphrase of a default account.

Parameters:

- address(Buffer): The address of a default delegate.

Usage

```
const { getPassphraseFromDefaultConfig } = require('lisk-framework/src/testing');
const address = cryptography.getAddressFromLisk32Address('lskrj5kmabsxk8z3wnkk5gm9jsqfbtqqvtmrg7mth');
const passphrase = getPassphraseFromDefaultConfig(address);
```

- Returns

getValidatorFromDefaultConfig

Returns the hash onion for a default delegate.

Parameters:

1. address(Buffer): The address of a default delegate.
2. count(number).

Usage

```
const { getValidatorFromDefaultConfig } = require('lisk-framework/src/testing');
const address = cryptography.getAddressFromLisk32Address('lskrj5kmabsxk8z3wnkk5gm9jsqfbtqqvtmrg7mth');
const validator = getValidatorFromDefaultConfig(address, 1);
```

- Returns

Mocks

StateStoreMock

Returns a mock for the state store.

Usage

```
const { StateStoreMock } = require('lisk-framework/src/testing');
```

- Returns

Make the full implementation of the @liskhq/lisk-db library.

Parameters:

- MockInputObject: Input data for the state store mock, (this is optional).

Example

```
const stateStore = new StateStoreMock({
  data: [
    { key: Buffer.from('0a0101', 'hex'), value: Buffer.from('0a0a6c69736b206973206f6b', 'hex') },
    { key: Buffer.from('0a0102', 'hex'), value: Buffer.from('0a0a6c69736b206973206f6b', 'hex') },
  ],
  isCacheEnabled: true,
});
```

Usage

```
const { createMock } = require('lisk-framework/src/testing');
const account = cryptography.utils.getRandomBytes(20);
const stateStore = new createMock.StateStoreMock({
  data: [{ key: account, value: Buffer.from('0a0a6c69736b206973206f6b', 'hex') }],
});
```

channelMock

Returns a mock for the channel.

Usage

```
const { createMock } = require('lisk-framework/src/testing');
const channel = createMock.channelMock();
```

- Returns

consensusMock

Returns a mock for consensus functions.

Usage

```
const { createMock } = require('lisk-framework/src/testing');
const consensus = createMock.consensusMock();
```

- Returns

DataAccessMock

Returns a mock for the data access.

Parameters:

- opts(object): Input data for the data access mock, (this is optional).

```
const dataAccess = new createMock.DataAccessMock({
  db: new InMemoryKVStore(),
  chainModules: MockInputObject,
});
```

Usage

```
const { createMock } = require('lisk-framework/src/testing');
const dataAccess = new createMock.DataAccessMock();
```

loggerMock

Returns a mock of the logger.

Usage

```
const { createMock } = require('lisk-framework/src/testing');
const logger = createMock.loggerMock();
```

- Returns

reduceHandlerMock

Returns a mock for the reduce handler.

Usage

```
const { createMock } = require('lisk-framework/src/testing');
const reduceHandler = createMock.reduceHandlerMock();
```

- Returns

Functions

getBlockProcessingEnv

Used for integration testing.

Parameters:

```
const env = await getBlockProcessingEnv({
  options: {
    genesis: {
      block: {
        fromFile: path.resolve(__dirname, '../../../config/devnet/genesis_block.blob'),
      },
    },
  },
});
```

Usage

```
const { getBlockProcessingEnv } = require('lisk-framework/src/testing');
const env = await getBlockProcessingEnv();
```

- Returns

createDefaultApplicationEnv

Used for functional testing.

Parameters:

- appConfig(object): Environment variable for the application.

Make the full implementation of the @liskhq/lisk-db library.

```
const appConfig = {
  genesis: {
    block: {
      fromFile: path.resolve(__dirname, '../../../config/devnet/genesis_block.blob'),
    },
  },
};
```

Usage

```
const { createDefaultApplicationEnv } = require('lisk-framework/src/testing');
const env = await createDefaultApplicationEnv({ appConfig });
```

- Returns

createTransaction

Parameters:

```
const tx = createTransaction({
  module: 'token',
  command: 'transfer',
  fee: BigInt(100000000),
  params: {
    tokenID: Buffer.from('0000000000000000', 'hex'),
    amount: BigInt(100000000),
    recipientAddress: Buffer.from('ecd71870d1963316a97e3ac3408cf22a3d375ebf', 'hex'),
    data: 'transfer data',
  },
  passphrase: 'wagon stock borrow episode laundry kitten salute link globe zero feed marble',
  keys: {
    mandatoryKeys: [Buffer.from('0b4a9ff5ed6a3e7941a7c392ab9be23501183ea7a4a5ad2cd114eb74d03e1f3d', 'hex')],
    optionalKeys: [],
  },
});
```

Usage

```
const { createTransaction } = require('lisk-framework/src/testing');
const tx = createTransaction();
```

- Returns

createGenesisBlock

Parameters:

```
const { createGenesisBlock } = require('lisk-framework/src/testing');
const genesisBlock = createGenesisBlock({
  modules: [TokenModule, AnotherModule],
  accounts: [{ address: Buffer.from('0a0a6c69736b206973206f6b', 'hex') }],
  keysList: [{ address: Buffer.from('0a0a6c69736b206973206f6b', 'hex') }],
  initDelegates: [Buffer.from('0a0a6c69736b206973206f6b', 'hex')],
  initRounds: 3,
  chainID: Buffer.from('10000000', 'hex'),
});
```

Usage

```
const { createGenesisBlock } = require('lisk-framework/src/testing');
const genesisBlock = createGenesisBlock();
```

- Returns

getModuleInstance

Parameters:

```
const { getModuleInstance } = require('lisk-framework/src/testing');
const moduleInstance = getModuleInstance(
  Module,
  {
    genesisConfig: GenesisConfig,
    options: ModuleOptionObject,
    logger: LoggerMock,
  },
);
```

Usage

```
const { getModuleInstance } = require('lisk-framework/src/testing');
const moduleInstance = getModuleInstance(TokenModule);
```

- Returns

getAccountSchemaFromModules

Parameters:

```
const { getAccountSchemaFromModules } = require('lisk-framework/src/testing');
const accountSchema = getAccountSchemaFromModules([ new TokenModule() ]);
```

Usage

```
const { getAccountSchemaFromModules } = require('lisk-framework/src/testing');
const accountSchema = getAccountSchemaFromModules([ new TokenModule() ]);
```

- Returns

createBlock

Parameters:

```
const { createBlock } = require('lisk-framework/src/testing');
const block = createBlock({
  header: {
    version: 2,
    timestamp: 1000,
    previousBlockID: Buffer.from('0a0a6c69736b206973206f6b', 'hex'),
    height: 1,
    generatorAddress: Buffer.from('0a0a6c69736b206973206f6b', 'hex'),
  },
  payload: [],
});
```

Usage

```
const { createBlock } = require('lisk-framework/src/testing');
const block = createBlock({
  header: {
    previousBlockID: Buffer.from('0a0a6c69736b206973206f6b', 'hex'),
    height: 1,
    generatorAddress: Buffer.from('0a0a6c69736b206973206f6b', 'hex'),
  },
});
```

- Returns

encodeBlockHeader

Parameters:

- header: Block header.
- skipSignature(boolean): signature will be skipped if set as TRUE, (this is optional).

Usage

```
const { encodeBlockHeader } = require('lisk-framework/src/testing');
const encodedBlockHeader = encodeBlockHeader(blockHeader);
```

- Returns

createBlockHeaderWithDefaults

Creates a default block header.

Parameters:

- header: Block header partial, (this is optional).

Usage

```
const { createBlockHeaderWithDefaults } = require('lisk-framework/src/testing');
const blockHeader = createBlockHeaderWithDefaults();
```

- Returns

createFakeBlockHeader

Creates a block header with a fake signature and ID.

Parameters:

- header: Block header partial, (this is optional).

Usage

```
const { createFakeBlockHeader } = require('lisk-framework/src/testing');
const blockHeader = createFakeBlockHeader();
```

- Returns

createNewGenesisBlockApplyContext

Parameters:

```
const { createNewGenesisBlockApplyContext } = require('lisk-framework/src/testing');
const applyContext = createNewGenesisBlockApplyContext(stateStore, {
  header: genesisBlock.header,
  assets: genesisBlock.assets,
});
```

Usage

```
const { createNewGenesisBlockApplyContext } = require('lisk-framework/src/testing');
const applyContext = createNewGenesisBlockApplyContext(stateStore, genesisBlock);
```

- Returns

createBeforeBlockApplyContext

Parameters:

```
const { createBeforeBlockApplyContext } = require('lisk-framework/src/testing');
const beforeBlockApplyContext = createBeforeBlockApplyContext({
  header: genesisBlock.header,
  assets: genesisBlock.assets,
  stateStore,
  eventQueue: new EventQueue(),
  chainID: Buffer.from('10000000', 'hex'),
});
```

Usage

```
const { createBeforeBlockApplyContext } = require('lisk-framework/src/testing');
const beforeBlockApplyContext = createBeforeBlockApplyContext();
```

- Returns

createNewBlockApplyContext

Parameters:

```
const { createNewBlockApplyContext } = require('lisk-framework/src/testing');
const newBlockApplyContext = createNewBlockApplyContext(stateStore, {
  header: genesisBlock.header,
  assets: genesisBlock.assets,
});
```

Usage

```
const { createNewBlockApplyContext } = require('lisk-framework/src/testing');
const newBlockApplyContext = createNewBlockApplyContext(stateStore, genesisBlock);
```

- Returns

createTransactionApplyContext

Parameters:

```
const { createTransactionApplyContext } = require('lisk-framework/src/testing');
const transactionApplyContext = createTransactionApplyContext({
  transaction: tx,
  stateStore,
  eventQueue: new EventQueue(),
  chainID: Buffer.from('10000000', 'hex'),
});
```

Usage

```
const { createTransactionApplyContext } = require('lisk-framework/src/testing');
const transactionApplyContext = createTransactionApplyContext();
```

- Returns

createApplyAssetContext

Parameters:

```
const { createApplyAssetContext } = require('lisk-framework/src/testing');
const applyAssetContext = createApplyAssetContext({
  transaction: tx,
  stateStore,
  eventQueue: new EventQueue(),
  chainID: Buffer.from('10000000', 'hex'),
});
```

Usage

```
const { createApplyAssetContext } = require('lisk-framework/src/testing');
const applyAssetContext = createApplyAssetContext();
```

- Returns

createValidateAssetContext

Parameters:

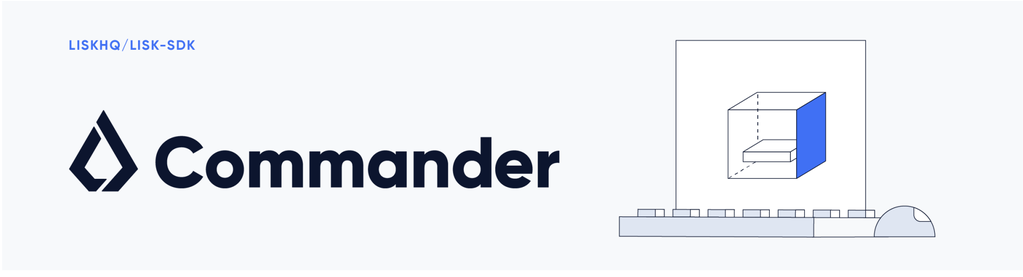
```
const { createValidateAssetContext } = require('lisk-framework/src/testing');
const validateAssetContext = createValidateAssetContext({
  transaction: tx,
  chainID: Buffer.from('10000000', 'hex'),
});
```

Usage

```
const { createValidateAssetContext } = require('lisk-framework/src/testing');
const validateAssetContext = createValidateAssetContext();
```

- Returns

What is Lisk Commander?



Lisk Commander is a command line interface tool designed to be a low entry point, coupled with being the most convenient and user-friendly tool for developers who are looking for an efficient way of coding.

Lisk Commander allows communication from the command line with a remote or local node in order to access and utilise Lisk-related functionalities.

What can be achieved with Lisk Commander?

- Bootstrapping a blockchain application.
- Decrypting/encrypting passphrases or messages.
- Creating new accounts.
- And much more...For further details please see the command reference.

Table of Contents

Setup

- Dependencies
- Pre-installation
- Installation

Usage

Upgrade

Setup

> Note
>
> Supported platforms: Platforms with Node.js versions 8.3 and above.

Dependencies

The following dependencies need to be installed to run the Lisk SDK:

Dependencies	Version
Node.js	v16.15.0

Pre-installation

Install Node.js

Node.js is supported on most operating systems. Follow the instructions for your operating system on the Node.js downloads page. Lisk Commander requires Node.js versions 8.3 or above.

NPM is automatically installed along with Node.js.

Verify Node.js installation

Confirm that Node.js and NPM have been successfully installed by executing the following command:

```
node --version
npm --version
```

Installation

To install the latest version of Lisk Commander as a global command, install the NPM package as shown below:

```
npm install --global lisk-commander
```

Upon successful completion, NPM will add the Lisk Commander executable to your PATH.

Usage

> Tip
>
> For a full list of all commands and their available options, see the Command reference

Pass arguments and options directly from the command line, as shown in the example below:

```
$ lisk (-v|--version|version) # returns the current version of Lisk Commander.
$ lisk [TOPIC]:[ACTION] # to run a command. Available actions depend on the command.
$ lisk [TOPIC] # displays help for a specific command.
$ lisk # displays general help and the command list.
```

```
$ lisk
A command line interface for Lisk

VERSION
  lisk-commander/5.1.10 darwin-x64 node-v16.15.0

USAGE
  $ lisk [COMMAND]

TOPICS
  account     Commands relating to Lisk accounts.
  generate    Commands relating to Lisk generator.
  message     Commands relating to user messages.
  passphrase  Commands relating to Lisk passphrases.

COMMANDS
  autocomplete        display autocomplete installation instructions
  console             Lisk interactive REPL (Read-eval-print loop), session to run commands.
  hash-onion          Create hash onions to be used by the forger.
  help                display help for lisk
  init                Bootstrap a blockchain application using Lisk SDK.
  network-identifier
```

See concrete examples on the Commands page.

Upgrade

To update your global installation to the latest version of Lisk Commander, simply execute the following command:

```
npm update --global lisk-commander
```

Commands

Table of Contents

General commands
- Help
- Init
- Console
- Hash-onion
- Network identifier

Account
- Create account
- Show account

Generate
- Generate wasm
- Generate module
- Generate plugin

Message
- Decrypt message
- Encrypt message
- Sign message
- Verify message

Passphrase
- Decrypt passphrase
- Encrypt passphrase

General commands

Help

Displays command reference.

```
$ chainfire help
Chainfire command-line tool

Usage:
  chainfire [command]

Available Commands:
  help        Help about any command
  init        Bootstrap a blockchain application using the Lisk SDK

Flags:
  -h, --help   help for chainfire

Use "chainfire [command] --help" for more information about a command.
```

Init

Bootstrap a blockchain application using the Lisk SDK.

```
$ chainfire init --help
Bootstrap application skeleton.

Usage:
  chainfire init <application_name> [flags]

Arguments:
  <application_name>  (required) Name to create the project.

Flags:
  -h, --help                help for init
      --template string     Template to bootstrap the application. It will run from `~/.lisk/commander/templates/<template>`. Use `--template=default` to use the default template. (default "default")
      --version string      Version of the Lisk SDK to use to bootstrap the application. (default "latest")

Examples:
$ chainfire init my_app
$ chainfire init my_app --template=default
$ chainfire init my_app --version=5.0.0
$ chainfire init my_app --template=./templates/my_template
```

Console

Lisk interactive REPL session to run commands.

```
$ chainfire console --help
Start interactive REPL session

Usage:
  chainfire console [flags]

Flags:
  -d, --data-path string   Set custom path for blockchain data. (default "~/.lisk/chainfire")
  -h, --help               help for console
  -p, --port string        Port of IPC to connect to. (default "7887")
  -s, --socket-path string Set custom path for IPC socket.

Global Flags:
      --no-color   Disable colors in output
```

Hash-onion

Creates a hash-onion output to be used by a Forger.

```
$ chainfire hash-onion --help
Creates hash onion to be used by the Forger.

Usage:
  chainfire hash-onion [flags]

Flags:
  -c, --count count   Number of hashes to produce. (default 1000000)
  -d, --distance int  Distance between hashes in the onion. (default 2000)
  -h, --help          help for hash-onion
  -o, --output file   Write output to specified file instead of stdout.

Global Flags:
      --no-color   Disable colors in output
```

Network identifier

Displays the Network identifier based on the passed hash of the genesis block of the network. The Network identifier is used to distinguish between blocks/transactions intended for that network. In this way, signatures created within a network can not be used in exactly the same way (i.e. replayed) on another network.

```
$ chainfire network identifier --help
Displays the Network identifier based on the passed hash of the genesis block of the network.

Usage:
  chainfire network identifier [genesis_block_id] [flags]

Arguments:
  [genesis_block_id]  ID of genesis block from the network

Flags:
  -h, --help                help for network identifier
      --output-public-key   Print public key in JSON format. The key must be a valid hexadecimal string.
      --pretty              Prints JSON in pretty format rather than condensed. Has no effect if not in JSON. Has no effect on the default behaviour or when printing to a file.

Global Flags:
      --no-color   Disable colors in output

Examples:
# --> Network identifier for the given genesis block or and community identifier.
$ chainfire network identifier 4a48e6c0f0077ba2a7b6c1d4a7be6ed7b3466fe7f62d9189d2a7bdf7f1a2b03f

Example JSON output
{"networkIdentifier":"4a48e6c0f0077ba2a7b6c1d4a7be6ed7b3466fe7f62d9189d2a7bdf7f1a2b03f"}
```

Autocomplete

Display instructions how to enable auto-completion for Lisk Commands commands.

```
$ lisk autocomplete --help
Display instructions how to enable auto-completion for Lisk Commands commands.

Usage:
  lisk autocomplete [flags]

Arguments:
  SHELL  shell type

Options:
      --refresh-cache  Refresh cache (ignores displaying instructions)

Examples:
$ lisk autocomplete
$ lisk autocomplete bash
$ lisk autocomplete zsh
$ lisk autocomplete --refresh-cache
```

Account

Commands relating to Lisk accounts.

```
$ lisk account --help
Commands relating to Lisk accounts.

Usage:
  lisk account [command]

Available Commands:
  create      Create account. Generates a passphrase with 32 corresponding public/private key pair and Lisk address.
  show        Show account information for a given passphrase.

Global Flags:
      --no-color   Disable colors in output
```

Create account

The returns a randomly generated mnemonic passphrase with its corresponding public/private key pair and Lisk address.

```
$ lisk account create --help
Create account. Generates a passphrase with 32 corresponding public/private key pair and Lisk address.

Usage:
  lisk account create [number] [flags]

Arguments:
  [number]  (optional, default=1) Number of accounts to create.

Flags:
      --output-public-key   Print public key in JSON format. The key must be a valid hexadecimal string.
      --pretty              Prints JSON in pretty format rather than condensed. Has no effect if not in JSON. Has no effect on the default behaviour or when printing to a file.

Global Flags:
      --no-color   Disable colors in output

Examples:
$ lisk account create
$ lisk account create 10
```

Example output

```
{
  "passphrase": "lizard white stamp hockey red bike fish dash survey story capital",
  "privateKey": "4ce0ab2d9393e8745a70420f337cab802ef5d4774e7a9c2cee3201f285a0e719ad1ca2033103ed0fd1c5c3deb8fcb9c725ab1769a6d6d7ec38fd54941ef2d5b0db",
  "publicKey": "ad1ca2033103ed0fd1c5c3deb8fcb9c725ab1769a6d6d7ec38fd54941ef2d5b0db",
  "address": "lskdsmpbyc6wppjszm5k9ddkkbbmqmjpkwpkkqhq"
}
```

Show account

Shows account information for a given passphrase. It displays the Lisk address, the address as hexadecimal string, the publickey and the privatekey (for testing or the advanced developer).

```
$ lisk account show --help
Show account information for a given passphrase.

Usage:
  lisk account show [flags]

Flags:
  -p, --passphrase=passphrase  Specifies a source for your secret passphrase.
                               - Environment variable: set your passphrase to an env variable and pass it to the command (e.g. --passphrase=env:MY_PASSPHRASE)
                               - Command: pass a command to your command (e.g. --passphrase=cmd:"cat ~/.mypassphrase")
                               - Direct: pass your passphrase directly to the command (not recommended) (default "env:LISK_PASSPHRASE")
      --public-key             Include the public key in the output. This option is provided for the convenience of the developer.

Global Flags:
      --no-color   Disable colors in output

Example:
$ chainfire show
```

Example output

```
{
  "address": "lskdsmpbyc6wppjszm5k9ddkkbbmqmjpkwpkkqhq",
  "binaryAddress": "ad1ca2033103ed0fd1c5c3deb8fcb9c725ab1769",
  "publicKey": "ad1ca2033103ed0fd1c5c3deb8fcb9c725ab1769a6d6d7ec38fd54941ef2d5b0db",
  "privateKey": "4ce0ab2d9393e8745a70420f337cab802ef5d4774e7a9c2cee3201f285a0e719ad1ca2033103ed0fd1c5c3deb8fcb9c725ab1769a6d6d7ec38fd54941ef2d5b0db"
}
```

Generate

Commands to auto generate different components of a blockchain application.

```
$ lisk generate --help
Commands to auto generate different components of a blockchain application.

Usage:
  lisk generate [command]

Available Commands:
  block       Generate a block skeleton for the given public key, name and id.
  module      Generate a module skeleton for the given name and id.
  plugin      Generate a plugin skeleton for the given name and id.

Global Flags:
      --no-color   Disable colors in output
```

Generate block

Creates an asset skeleton for the given module name, asset name, and ID.

```
$ lisk generate block --help
Generate a block skeleton for the given public key, name and id.

Usage:
  lisk generate block [public_key] [name] [id] [flags]

Arguments:
  [public_key]  Public key of the block
  [name]        Name of the block
  [id]          ID of the block

Flags:
  -h, --help                help for block
      --template string     Template to bootstrap the application. It will run from `~/.lisk/commander/templates/<template>`. Use `--template=./my_template` to use a custom template. (default "default")

Global Flags:
      --no-color   Disable colors in output

Example:
$ chainfire generate block 0202c3d4e5 1000 2000
```

Generate module

Creates a module skeleton for the given name and ID.

```
$ lisk generate module --help
Generate a module skeleton for the given name and ID.

Usage:
  lisk generate module [name] [id] [flags]

Arguments:
  [name]  Module name.
  [id]    Module id. Should be atleast 1000 or more

Flags:
  -h, --help                help for module
      --template string     Template to bootstrap the application. It will run from `~/.lisk/commander/templates/<template>`. Use `--template=./my_template` to use a custom template. (default "default")

Global Flags:
      --no-color   Disable colors in output

Example:
$ chainfire generate module MODULENAME MODULEID
```

Generate plugin

Creates a custom plugin for the given alias.

```
$ lisk generate plugin --help
Generate a plugin skeleton for the given alias.

Usage:
  lisk generate plugin [alias] [flags]

Arguments:
  [alias]  Name of the plugin.

Flags:
  -h, --help                help for plugin
      --template string     Path to create the plugin
                            Template to bootstrap the application. It will run from `~/.lisk/commander/templates/<template>`. Use `--template=./my_template` to use a custom template. (default "default")

Global Flags:
      --no-color   Disable colors in output

Example:
$ chainfire generate plugin myplugin
$ chainfire generate plugin myplugin --template ./my_template
```

Message

Commands relating to user messages.

```
$ lisk message --help
Commands relating to user messages.

Usage:
  lisk message [command]

Available Commands:
  decrypt     Decrypt message
  encrypt     Encrypt message
  sign        Sign message
  verify      Verify message

Global Flags:
      --no-color   Disable colors in output
```

Decrypt message

Decrypts a previously encrypted message from a given sender public key for a known nonce using your secret passphrase.

Important

Important: As the secret passphrase is a sensitive input, it can be provided using one of the methods described in the Secret input section. The encrypted message can be provided either directly as an argument, or by specifying a source with the `--message` option. If both the secret passphrase and the encrypted message are provided via stdin, the secret passphrase must be provided in the first line, and the encrypted message must be provided in the subsequent lines.

```
$ lisk message decrypt --help
Decrypts a previously encrypted message from a given sender public key for a known nonce using your secret passphrase.

Usage:
  lisk message decrypt [options] [message] [nonce] [senderPublicKey]

Arguments:
  [message]          Encrypted message to decrypt.
  [nonce]            Nonce which was provided during encryption.
  [senderPublicKey]  Public key of the sender of the message.

Options:
  -m, --message=message        Specifies a source for providing a message to the command. If a string is provided directly as an argument, this option will be ignored. Otherwise, this option can be used to specify where to read the message from. (e.g. --message=./path/to/file.txt)
                               - Environment: specify environment variable that stores the message (e.g. --message=env:MESSAGE)
                               - File: specify path to file that contains the message (e.g. --message=file:./path/to/file.txt)
                               - Direct: pass the message directly to the command (e.g. --message="Hello World")
  -p, --passphrase=passphrase  Specifies a source for your secret passphrase. Command will prompt you for input if this option is not provided.
                               - Environment: set your passphrase to an env variable and pass it to the command (e.g. --passphrase=env:MY_PASSPHRASE)
                               - Command: pass a command to your command (e.g. --passphrase=cmd:"cat ~/.mypassphrase")
                               - Direct: pass your passphrase directly to the command (not recommended) (default "env:LISK_PASSPHRASE")
      --pretty                 Prints JSON in pretty format rather than condensed. Has no effect if not in JSON. Has no effect on the default behaviour or when printing to a file.

Global Flags:
      --no-color   Disable colors in output

Example:
$ chainfire decrypt 0a0b1c2d3e4f5a6b7c8d9e0f1a2b3c4d5e6f7a8b9c0d1e2f3a4b5c6d7e8f9
```

Sensitive inputs

Table of Contents

Via a password prompt

Via an environmental variable

Via a file

Via stdin

Via a plaintext option

There are certain instances whereby Lisk Commander requires sensitive inputs such as passphrases or secret messages, which you may not wish to include directly as command line arguments due to the fact that they will be visible. For example, in the bash history or process managers. This section describes the various options available for securely including sensitive inputs within Lisk Commander commands. The format for providing Lisk Commander with sensitive inputs is loosely based on the OpenSSL format.

The `message:encrypt` command is used for the given examples with a secret passphrase as a sensitive input, however the same pattern applies to other commands which also accept sensitive inputs.

Via a password prompt

If a passphrase is required for a command, and no source is provided via the `--passphrase` option, Lisk Commander will initiate a prompt for a password input, so the required passphrase can be typed in directly. Please note that this approach is only available for passphrases, and not other kinds of sensitive inputs.

```
$ lisk message:encrypt bba7e2e6a4639c431b68e31115a71ffefcb4e025a4d1656405dfdcd8384719e0 'Hello world'
Please enter your secret passphrase: ****************************************************************
Please re-enter your secret passphrase: ****************************************************************
```

Via an environmental variable

If your passphrase is stored in an environmental variable, it is possible to specify the variable name using the `–passphrase` option. Please be aware that this approach is only available for passphrases, and not other kinds of sensitive inputs.

```
lisk message:encrypt bba7e2e6a4639c431b68e31115a71ffefcb4e025a4d1656405dfdcd8384719e0 'Hello world' --pass
```

Via a file

If your passphrase is stored in a file, it is possible to specify the path using the –passphrase option, and Lisk Commander will read the passphrase from the first line. If other types of data are stored in a file, it is possible to specify the path using the –data option, and Lisk Commander will read the entire contents of the file.

```
lisk message:encrypt bba7e2e6a4639c431b68e31115a71ffefcb4e025a4d1656405dfdcd8384719e0 --data file:./secret
```

Via stdin

Lisk Commander will read a passphrase from the first line of stdin via the `–passphrase` option, or multiple lines for other sorts of data via the `–data` option. If stdin is specified as the source of both the passphrase and other additional data, then the passphrase is assumed to be the first line, and the data will follow. Please note that the actual use of echo with plaintext strings shown in the example below is **not recommended**:

```
$ echo 'my secret passphrase on the first line\Followed by\nsome data that\nspans multiple lines' | lisk m
```

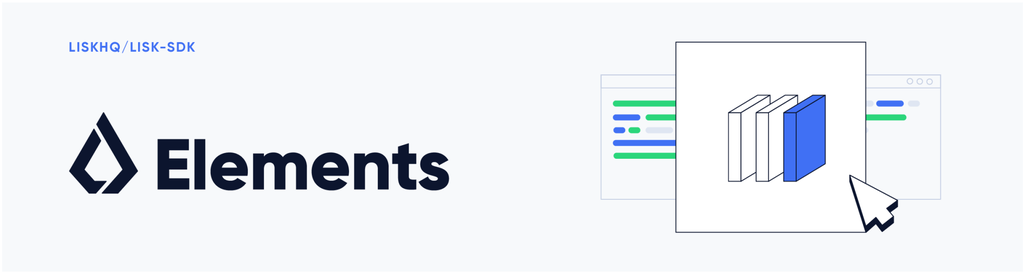
Via a plaintext option

⚠ Warning

For convenience, Lisk Commander also supports providing a plaintext passphrase via the `–passphrase` option. This can be useful for prototyping or testing, (for example on the testnet). However, this **should never be used when security is a concern**. Please note that this approach is only available for passphrases, and not other kinds of sensitive inputs.

```
lisk message:encrypt bba7e2e6a4639c431b68e31115a71ffefcb4e025a4d1656405dfdcd8384719e0 'Hello world' --pass
```

What is Lisk Elements?



Lisk Elements is a collection of JavaScript libraries, each tackling a certain topic of blockchain and Lisk-related functionalities.

Table of Contents

Package list

- Used in frontend
- Used in backend

Setup

- Dependencies
- Pre-installation
- Installation

Usage

- Node.js
- Browser

Update

Package list

Used in frontend

Name	NPM	Specification
Client	@liskhq/lisk-client	lisk-client

> **Note**
>
> The `Client` package is specifically designed to be used in the browser and other frontends. It is a collection of the following packages:
>
> `apiClient`, `codec`, `cryptography`, `passphrase`, `transactions`, `utils`, `tree` and `validator`.

Used in backend

Name	NPM	Specification
API client	@liskhq/lisk-api-client	lisk-api-client
BFT	@liskhq/lisk-bft	lisk-bft
Chain	@liskhq/lisk-chain	lisk-chain
Codec	@liskhq/lisk-codec	lisk-codec
Cryptography	@liskhq/lisk-cryptography	lisk-cryptography
DB	@liskhq/lisk-db	lisk-db
Genesis	@liskhq/lisk-genesis	lisk-genesis
P2P	@liskhq/lisk-p2p	lisk-p2p
Passphrase	@liskhq/lisk-passphrase	lisk-passphrase
Transaction pool	@liskhq/lisk-transaction-pool	lisk-transaction-pool
Transactions	@liskhq/lisk-transactions	lisk-transactions
Tree	@liskhq/lisk-tree	lisk-tree
Utils	@liskhq/lisk-utils	lisk-utils
Validator	@liskhq/lisk-validator	lisk-validator

Setup

Dependencies

The following dependencies need to be installed to run the Lisk SDK:

Dependencies	Version
Node.js	v16.15.0

Pre-installation

Lisk Elements requires Node.js as the underlying engine for code execution. This document describes how to install Node.js and NPM for installation via NPM.

Install node

Follow the instructions for your operating system on the Node.js downloads page.

> **Note**
>
> NPM is automatically installed along with Node.js.

Verify installation

Confirm that Node.js and NPM have been successfully installed by running the following commands:

```
node --version
npm --version
```

Installation

This section details how to install Lisk Elements for a given distribution.

> **Note**
>
> As all packages in Lisk Elements are independent from each other, they can be installed seperately.

Installation via NPM

To install the latest version of Lisk Elements for use as a dependency in a Node.js project, please proceed with the following:

Complete library

```
# Complete Lisk SDK
npm install lisk-sdk
# Lisk Elements packages only
npm install lisk-elements
# Frontend related packages only
npm install @liskhq/lisk-client
```

Specific package

```
npm install @liskhq/<package-name> (1)
```

(1) `<package-name>` can be any of the packages in Lisk Elements.

Upon successful completion, NPM will add the Lisk Elements package to your `package.json` file.

Load via CDN (Content Delivery Network)

Include the script below using the following HTML. The `lisk` variable will be exposed.

Client library

```
<script src="https://js.lisk.com/lisk-client-5.2.1.js"></script>
```

Or minified:

```
<script src="https://js.lisk.com/lisk-client-5.2.1.min.js"></script>
```

To include other packages of Lisk Elements, replace `lisk-client` with any of the packages of Lisk Elements.

Usage

Node.js

Simply import, (or require) the package and access its functionality according to the relevant namespace.

Example with the client package:

```
import lisk from '@liskhq/lisk-client';
//or
const lisk = require('@liskhq/lisk-client');
```

Browser

Load the Lisk Elements script via the CDN. For example, to load the minified version 5.2.1 of Lisk Elements, include the following script which will then expose the `lisk` variable:

```
<script src="https://js.lisk.com/lisk-client-5.2.1.min.js"></script>
<script>
    const api = lisk.apiClient.createWSClient('ws://localhost:8080/ws');
</script>
```

Update

To update your installation to the latest version of Lisk Elements, simply run the following command:

Complete library

```
npm update lisk-elements
```

Specific package

```
npm update @liskhq/<package-name> (1)
```

API client

Table of Contents

Installation

Upgrade

- APIClient interface

The Lisk Elements API Client provides a convenient wrapper for interacting with the public API of nodes on the Lisk network.

> **Note**
>
> Please use the `apiClient` of the `lisk-client package` for all code that runs in the browser

Installation

This adds Lisk Client as a dependency of your project:

```
$ npm install @liskhq/lisk-api-client
```

Upgrade

```
npm update @liskhq/lisk-api-client
```

APIClient interface

```
interface APIClient {
  public disconnect() => Promise<void>;
  public async invoke<T>(actionName: string, params?: Record<string, unknown>): Promise<T>
  public subscribe(eventName: string, cb: EventCallback): void
  public transaction: {
    get: (id: Buffer) => Promise<Record<string, unknown>>,
    getFromPool: () => Promise<Record<string, unknown>[]>,
    create: (
      input: {
        moduleID?: number; // id takes priority
        moduleName?: number;
        assetID?: number; // id takes priority
        assetName?: number;
        fee: bigint;
        nonce: bigint;
        senderPublicKey: Buffer;
        asset: Record<string, unknown>;
        signatures?: Buffer[];
      },
      passphrase: string,
      options?: {
        nonce?: bigint;
        includeSenderSignature?: boolean;
        multisignatureKeys?: {
          mandatoryKeys: Buffer[];
          optionalKeys: Buffer[];
        };
      },
    ) => Promise<Record<string, unknown>>;
    sign: (transaction: Record<string, unknown>, passphrases: string[], options?: {
      includeSenderSignature?: boolean;
      multisignatureKeys?: {
        mandatoryKeys: Buffer[];
        optionalKeys: Buffer[];
      };
    }) => Promise<Record<string, unknown>>;
    send: (input: Record<string, unknown>) => Promise<void>;
    getMinFee: (tx: Record<string, unknown>) => bigint;
    encode: (input: Record<string, unknown>) => Buffer;
    decode: (input: Buffer) => Record<string, unknown>;
    toJSON(transaction: Record<string, unknown>) => Record<string, unknown>;
    fromJSON(transaction: Record<string, unknown>) => Record<string, unknown>;
  }
  public node: {
    getNodeInfo: () => Promise<Record<string, unknown>>;
    getNetworkStats: () => Promise<Record<string, unknown>>;
    getConnectedPeers: () => Promise<Peer[]>;
    getDisconnectedPeers: () => Promise<Peer[]>;
  }
  public block: {
    get: (id: Buffer) => Promise<Record<string, unknown>>;
    getByHeight: (height: number) => Promise<Record<string, unknown>>;
    encode: (input: Record<string, unknown>) => Buffer;
    decode: (input: Buffer) => Record<string, unknown>;
    toJSON(
              block: BlockType,
        ) => {
              header: Record<string, unknown>;
              payload: Record<string, unknown>[];
        };
        fromJSON(
              block: BlockType<string>,
        ) => {
              header: Record<string, unknown>;
              payload: Record<string, unknown>[];
        };
  }
  public account: {
    get: (address: Buffer) => Promise<Record<string, unknown>>;
    encode: (input: Record<string, unknown>) => Buffer;
    decode: (input: Buffer) => Record<string, unknown>;
    toJSON(account: Record<string, unknown>) => Record<string, unknown>;
    fromJSON(account: Record<string, unknown>) => Record<string, unknown>;
  }
}
```

Client

Table of Contents

Installation

Upgrade

Usage

- Creating an APIClient

The @liskhq/lisk-client package provides a default set of packages for use by clients of the Lisk network.

> **Note**
>
> The `Client` package is a collection of the following packages:
>
> api-client, codec, cryptography, passphrase, transactions, utils, tree, validator

Installation

To add Lisk client as a dependency of your project execute the following command:

```
$ npm install @liskhq/lisk-client
```

Upgrade

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-client
```

Usage

Import using ES6 modules syntax:

```
import lisk from '@liskhq/lisk-client';
```

Or using Node.js modules:

```
const lisk = require('@liskhq/lisk-client');
```

Or import specific namespaced functionality:

```
import { apiClient, transactions } from '@liskhq/lisk-client';
// or
const { apiClient, transactions } = require('@liskhq/lisk-client');
```

Creating an APIClient

```
const { apiClient } = require('@liskhq/lisk-client');

let clientCache;

export const getClient = async () => {
    if (!clientCache) {
        clientCache = await apiClient.createWSClient('ws://localhost:8080/ws');
    }
    return clientCache;
};

export const useClient = async () => {
    const client = await getClient();
    const blockAtHeight123 = await client.block.getByHeight(123);
    client.subscribe('pluginAlias:eventAlias', ({ data }) => {
      console.log(data.info);
    });
};
```

BFT

Table of Contents

Installation

Update

@liskhq/lisk-bft is byzantine fault tolerance implementation according to the Lisk protocol.

Installation

```
npm install @liskhq/lisk-bft
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-bft
```

Chain

Table of Contents

Installation

Update

State store and database mechanism

- State store
- Data access

How state changes are rolled back

@liskhq/lisk-chain implements blocks and state management that are used for block processing according to the Lisk protocol.

Installation

```
npm install @liskhq/lisk-chain
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-chain
```

State store and database mechanism

Lisk-chain manages the fundamental blockchain logic including the data schema of blocks, transactions, and accounts. This includes the following points listed below:

- Validating the block headers, such as `schema`, `previousBlockID` or block reward, without any consideration of the BFT rules.
- Validating the transaction schema and minimum fee requirement.
- Validating the block generator based on the given set of validators and the slots.
- Providing database access to the blockchain data.
- Saving and deleting block information.

State store

The state store is a data structure that holds temporary state while processing a block. The store is classified primarily based on 3 states - Accounts, Chain, and Consensus. The `account` store holds state per account, the `chain` store holds blockchain wide state information, and finally the `consensus` store holds the internal state related to the BFT.

- The state store exposes an interface to create, restore, and finalize snapshots.
- The `account` store is used mainly for registering delegates, keys and token transfers.
- The `chain` state store handles the delegate vote weights, total burnt fees, block headers of the last three rounds, network identifier, and rewards for the last block.
- The `consensus` store contains the finalized height, the BFT voting ledger, and a list of validator's information.

Data access

The data access module provides simplified access to the persistent data storage (blockchain database).

Advantages of data access

- The abstraction of an actual database engine allows the application to switch between different databases.
- The decoupling of business logic allows more flexibility to modify the data model without impacting the application layer.

Features of data access

- Encoding and decoding block headers, accounts, and transactions.
- Caching of blocks.
- Access to blocks, accounts, chain state, and transactions.
- Saving and deleting blocks.

The last 500 blocks are cached in the memory in order to improve block access performance. The block related queries check in the cache memory before querying the storage.

How state changes are rolled back

The data access module implements a diff mechanism to roll back the blockchain to its previous state. For each new block that is added to the chain, the system looks for possible state changes (create, update, delete). This is followed by storing the state transition for each store (account, chain, consensus), by creating a collection of created, updated and deleted keys and their corresponding values. Whenever the chain needs to delete a block it can refer to the stored diff for a particular height and apply it to regain the state.

Codec

Table of Contents

- Installation
- Update
- Usage
- Codec interface

The @liskhq/lisk-codec package implements a decoder and encoder using Lisk JSON (JavaScript Object Notation), schema according to the Lisk protocol.

Installation

```
npm install @liskhq/lisk-codec
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-codec
```

Usage

```
//import * as codecFunctions from '@liskhq/lisk-codec';
import { codec as codecFunctions } from 'lisk-sdk';
// Codec can be used in two ways:
// 1) Use exposed default instance
codecFunctions.codec.encode(schema, object);
// 2) Create a new instance
const newCodec = new codecFunctions.Codec();
newCodec.encode(schema, object);
```

Codec interface

```
interface Codec {
  // adds schema to memory to improve the performance
  addSchema(schema: Schema): boolean;
  // encode the JavaScript Object message to Buffer using the schema provided
  encode(schema: Schema, message: object): Buffer;
  // decode the Buffer message to JavaScript Object
  decode<T>(schema: Schema, message: Buffer): T;
  // decode the message input to JSON object
  decodeJSON<T>(schema: Schema, message: Buffer): T;
  // encode the JSON object message to Buffer using the schema provided
  encodeJSON(schema: Schema, message: object): Buffer;
  // Convert JavaScript object to JSON object using the schema provided
  toJSON<T = object>(schema: Schema, message: object): T;
  // Convert JSON object to JavaScript object using the schema provided
  fromJSON<T = object>(schema: Schema, message: object): T;
}
```

Cryptography

- Class: `crypto`
- Constants
- Static
- Legacy
- Instance
- Instance methods
- Methods for encrypting/decrypting messages
  - cipher.update([data], [input_encoding], [output_encoding])
  - cipher.final([output_encoding])
  - cipher.setAAD(buffer)
  - cipher.getAuthTag()
- Methods for computing/verifying digests
  - hash.update(data, [input_encoding])
  - hash.digest([encoding])
- Methods for computing/verifying signatures
  - sign.update(data)
  - sign.sign(private_key, [output_format])
  - verify.update(data)
  - verify.verify(object, signature, [signature_format])
- Methods for computing shared secrets
  - diffieHellman.generateKeys([encoding])
  - diffieHellman.computeSecret(other_public_key, [input_encoding], [output_encoding])
  - diffieHellman.getPrime([encoding])
  - diffieHellman.getGenerator([encoding])
  - diffieHellman.getPublicKey([encoding])
  - diffieHellman.getPrivateKey([encoding])
  - diffieHellman.setPublicKey(public_key, [encoding])
  - diffieHellman.setPrivateKey(private_key, [encoding])
- Methods for verifying/signing messages
  - ecdh.generateKeys([encoding], [format])
  - ecdh.computeSecret(other_public_key, [input_encoding], [output_encoding])
  - ecdh.getPrivateKey([encoding])
  - ecdh.getPublicKey([encoding], [format])
  - ecdh.setPrivateKey(private_key, [encoding])
- Methods for converting between formats
  - crypto.publicEncrypt(public_key, buffer)
  - crypto.privateDecrypt(private_key, buffer)

DB

Table of Contents

Installation

Update

The @liskhq/lisk-db package providing a database access layer that is a thin wrapper around RocksDB.

Installation

```
npm install @liskhq/lisk-db
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-db
```

Genesis

Table of Contents

- Installation
- Update
- Usage
- Methods
  - createGenesisBlock

`@liskhq/lisk-genesis` is a package which contains genesis block creation functions in accordance with the Lisk protocol.

Installation

```
npm install @liskhq/lisk-genesis
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-genesis
```

Usage

```
const { TokenModule, DPoSModule, genesis, configDevnet } = require('lisk-sdk');

// Get account schemas of the modules which are registered with the application
const token = new TokenModule(configDevnet.genesisConfig).accountSchema;
const dpos = new DPoSModule(configDevnet.genesisConfig).accountSchema;
// Add fieldNumber starting from 2. Field number 1 is assigned to address of the account
token.fieldNumber = 2;
dpos.fieldNumber = 3;

// Pack all account schemas in one big object
const accountAssetSchemas = {
  token,
  dpos
};

const delegates = [
    {
        address: Buffer.from('25d4c1ed2da24ba856b894c209188f70e0caab6e', 'hex'),
                token: { balance: BigInt(2874239947) },
                dpos: { delegate: { username: 'ef374a2e8fb9934ad1db0fd5346eb7' } }
    }
];

const accounts = [
    {
        address: Buffer.from('dedfbd4c6a1ad1ef0d7a95cd6907114548d8c6d8', 'hex'),
        token: { balance: BigInt(653021139) }
    }
];

const genesisBlockParams = {
        initDelegates: delegates.map(a => a.address),
        accounts: [...accounts, ...delegates],
        accountAssetSchemas,
};

const genesisBlock = genesis.createGenesisBlock(genesisBlockParams);

console.log(genesisBlock);
```

Methods

createGenesisBlock

Generates a genesis block based on the provided block parameters.

Syntax

```
createGenesisBlock(params)
```

Parameters

- `params` (object): The block parameters that shall be used to generate the genesis block.

The following parameters are available:

accountAssetSchemas(object)
: Required. The genesis block needs to contain all account asset schemas for all modules which are registered with the respective blockchain application. The different account asset schemas are all grouped together in one big object and added as `accountAssetSchemas` to the genesis block params. `accountAssetSchemas` is one of the most important parameters for generating a valid genesis block, so make sure it includes all required account asset schemas.

accounts(array)
: Required. List of initial accounts in the network. The minimum required here is the `address`, but it can have other properties like `balance`. Addresses are expected to be in `Buffer` format.

initDelegates(array)
: Required. List of initial delegate addresses used during the bootstrap period to forge blocks. Addresses are expected to be in `Buffer` format.

initRounds?(number)
: Optional. Number of rounds for the bootstrap period, the default is 3.

height?(number)
: Optional. Height of the genesis block.

timestamp?(number)
: Optional. Timestamp of the genesis block.

previousBlockID?(Buffer)
: Optional. Previous block ID. For example, this can be used in case of regenesis.

Return value

The generated genesis block object.

```
{
  header: {
    generatorPublicKey: <Buffer >,
    height: 0,
    previousBlockID: <Buffer >,
    reward: 0n,
    signature: <Buffer >,
    timestamp: 1619688502,
    transactionRoot: <Buffer e3 b0 c4 42 98 fc 1c 14 9a fb f4 c8 99 6f b9 24 27 ae 41 e4 64 9b 93 4c a4 95 
    version: 0,
    asset: { initRounds: 3, initDelegates: [Array], accounts: [Array] },
    id: <Buffer 3d 2a f6 c2 32 7b 63 65 21 06 4b 9e a6 3e af 21 c5 99 18 88 4d 3f 82 11 42 d9 5d 65 08 d8 a
  },
  payload: []
}
```

P2P

Table of Contents

- Installation
- Update
- Usage
- Architecture
  - P2P
  - PeerPool
  - PeerServer
  - Peer
  - PeerBook
- Actions
- Events
  - Available events
- Examples

@liskhq/lisk-p2p is a package for creating unstructured P2P networks. The package in abstraction enables peer discovery, network security, and reliability. A developer can leverage all the underlying specifications proposed in LIP 0004: Introduce robust peer selection and banning mechanism and use the package for building P2P use cases.

Installation

```
npm install --save @liskhq/lisk-p2p
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-p2p
```

Usage

To run a P2P node.

```
const { P2P } = require('@liskhq/lisk-p2p');

const p2p = new P2P({
        nodeInfo: {
                wsPort: 5001,
                networkId: '123xyz',
                protocolVersion: '1.1',
        },
});

const run = async () => {
        await p2p.start();
};

run()
        .then(() => console.log('P2P node has started'))
        .catch(err => {
                console.log(`Error occurred while running p2p node: ${err}`);
                p2p.stop();
                process.exit(1);
        });
```

It will start a node that is available on `5001` websocket port.

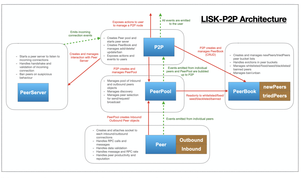
```
P2P node has started
```

Architecture

The architecture of the Lisk P2P library is divided into multiple parts (components or classes). The main parts of the P2P library are as follows:

- P2P
- PeerPool
- PeerServer
- PeerBook
- Peer

The responsibilities and interactions of these parts (components or classes) are summarized by the following diagram:



Before explaining the individual modules, it is helpful to become familiar with the following terminology:

nodeInfo
Contains the required data for establishing a successful connection, and is updated whenever a peer publishes new (nodeInfo) data.

peerInfo
Includes shared information which is the nodeInfo of a peer, and also the internal data accumulated on each peer by a node, related to productivity, reputation, rate limit, etc.

P2P

The P2P class is the main entry point of the P2P library that manages PeerPool, PeerBook, and PeerServer. Its responsibilities consist of the following:

- Expose the following actions:
  - getConnectedPeers(): Returns a list of all connected peers.
  - getDisconnectedPeers(): Returns a list of all disconnected peers.
  - applyPenaltyOnPeer(): Applies a penalty score to a specific peer for given peerId.
  - applyNodeInfo(): Updates node information.
  - getNodeInfo(): Returns node information.
  - start(): Starts PeerServer, loads previous peers and triggers discovery through seed nodes if not enough triedPeers.
  - stop(): Stops PeerServer, removes all the listeners and calls stop on PeerPool that eventually disconnects all the inbound/outbound connections.
- Expose actions and emit events bubbling up from Peer, PeerPool, and PeerServer.
- Setup all event listeners and corresponding event handlers.
- Handle getPeers remote procedure calls from peers.
- Manage CRUD (Create, Read, Update & Delete), operations on PeerBook that holds and manages all the peerInfos of all known peers.
- Loads previous peers and does the discovery. In case, if it doesn't have at least 100 triedPeers then it starts the discovery through seed peers provided in the config.

PeerPool

The PeerPool part manages all outbound/inbound connections, and maintains the number of required connections as per the configuration. Its responsibilities consist of the following:

- Trigger discovery through seedPeers.
- Create inbound and outbound peer objects.
- Contain all listeners and their corresponding handlers for each event coming out of Peer objects.
- Maintain the number of inbound, outbound, and also connections based on maxInboundConnections and maxOutboundConnections.
- Use selection functions to select peers to send or request.
- Provide security and priority for incoming connections based on net group, latency, response rate and connect time.
- Manage different trusted peer lists (whitelisted, fixed, blacklisted, seed), and their properties.
- Perform the periodic shuffling of outbound connections every 5 minutes.

PeerServer

The PeerServer part starts a server in order to listen for incoming connections. Its responsibilities are to perform the following:

- Provide a middleware to tackle attacks like PING/PONG, invalid events, etc.
- Immediately reject incoming connections that are blacklisted or banned.
- Check for duplicate connections and prevent the node from connecting to itself.
- Check for invalid payloads.
- Perform the handshake check for an incoming connection and create a valid peerInfo object for a peer.

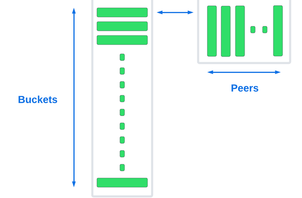
Peer

The Peer part is a class which is extended to have specific outbound and inbound classes for outgoing and incoming connections respectively. The outbound class is for outgoing connections which a node makes by creating a socket client, whereas the inbound class is for incoming connections. Please note that both inbound and outbound peers can be used for 2-way communication. However, a node chooses only outbound connections to make a request, and prioritizes outbound peers for sending information. The responsibilities for both type of connections are to perform the following:

- Create or assign a socket for communication on connection.
- Fetch the peer list and status of a peer on connection with outbound peers.
- Impose rate limits on messages and RPC (Remote-Procedure-Call), requests, and apply a penalty for misbehavior.
- Ping inbound peers for around 20-30 seconds to check for latency, and also to prove that the connection is alive, otherwise the connection is dropped.
- Emit all the peer related events including the ones coming from the network.

PeerBook

The PeerBook part is a directory of peers. It implements peer buckets for newPeers and triedPeers as described in LIP 0004. The newPeers and triedPeers are lists of peer buckets whereby each peer bucket consists of a list of 32 peers, and each peer bucket is chosen based on its IP prefix.



`newPeers` contains peers that either had no connection attempts or failed attempts. It consists of a list of 128 buckets with each bucket having 32 peers. Hence, the maximum number of peers that can be stored is 4096.

`triedPeers` contains the peers which a node was able to successfully establish a connection to, before or currently having an active connection. It has a list of 64 buckets with each bucket containing 32 peers. Therefore, the maximum number of peers that can be stored is 2048.

Both peer lists store peerInfo of a peer and have an eviction mechanism based on the time it resides in a bucket, or is evicted randomly when the bucket is full. In addition, they are also responsible for handling the movement of peers from triedPeers to newPeers and vice versa.

The PeerBook also manages static lists that are configured as described below:

- `seedPeers`: Peers that will be used to perform an initial discovery.
- `whitelistedPeers`: Peers that are always given a slot for incoming connections and where banning cannot be applied from outside the library.
- `fixedPeers`: Peers that a node establishes an outbound connection to, and never drops it. In the case whereby a disconnection does occur, it attempts to re-establish the connection after a certain period of time.
- `blacklistedPeers`: Peers that are blacklisted by the IP prefix and are not allowed to establish a connection.

Furthermore, the Peerbook maintains a list of banned peers and creates a timeout of 24 hours for each banned peer, after which it can be unbanned.

The PeerBook exposes actions to perform CRUD operations based on peerId whereby the caller is agnostic to newPeers, triedPeers, and their bucketing system.

Actions

It provides a simple interface to send, request, broadcast information, and many more functions to interact with the network.

- `p2p.start()`: start a P2P node after creating an instance.
- `p2p.stop()`: stop a P2P node.
- `p2p.config`: get the config of the node.
- `p2p.isActive`: check the status if the node is up and running.
- `p2p.nodeInfo`: check the node status and information.
- `applyNodeInfo(nodeInfo: P2PNodeInfo)`: broadcast updated `nodeInfo` to the network.
- `p2p.getConnectedPeers()`: get all the connected peers that are connected to your node in the network.
- `p2p.getDisconnectedPeers()`: get all the disconnected peers that are part of the network but not connected to your node.
- `p2p.request(packet: P2PRequestPacket)`: request information from the network that will run the peer selection and finds an appropriate peer for you to request information.
- `p2p.send(message: P2PMessagePacket)`: sends information to 16 connected peers chosen by the peer selection function.
- `p2p.broadcast(message: P2PMessagePacket)`: broadcast information to all the connected peers.
- `p2p.requestFromPeer(packet: P2PRequestPacket,peerId: string)`: request from a specific peer in the network.
- `p2p.sendToPeer(message: P2PMessagePacket, peerId: string)`: sends information to a specific peer in the connected peers.

Events

Allows to listen to various events on the network, to observe the network activities more closely and take the appropriate actions, if necessary.

```
// When a peer updates its information
p2p.on(EVENT_UPDATED_PEER_INFO, (peerInfo: P2PPeerInfo) => {
        // Take any action based peer update event
});
// When a peer sends any information
p2p.on(EVENT_MESSAGE_RECEIVED, (message: P2PMessagePacket) => {
        // Take any action based on message received
        const { event, data, peerId } = message;
});
// When a peer requests any information
p2p.on(EVENT_REQUEST_RECEIVED, async (request: P2PRequest) => {
        // Take any action based on request received and respond with `end(results)` with results or return
        const { procedure, data, peerId, end, error } = request;
});
```

Available events

- `EVENT_BAN_PEER`
- `EVENT_CLOSE_INBOUND`
- `EVENT_CLOSE_OUTBOUND`
- `EVENT_CONNECT_ABORT_OUTBOUND`
- `EVENT_CONNECT_OUTBOUND`
- `EVENT_DISCOVERED_PEER`
- `EVENT_FAILED_PEER_INFO_UPDATE`
- `EVENT_FAILED_TO_ADD_INBOUND_PEER`
- `EVENT_FAILED_TO_COLLECT_PEER_DETAILS_ON_CONNECT`
- `EVENT_FAILED_TO_FETCH_PEER_INFO`
- `EVENT_FAILED_TO_FETCH_PEERS`
- `EVENT_FAILED_TO_PUSH_NODE_INFO`
- `EVENT_FAILED_TO_SEND_MESSAGE`
- `EVENT_INBOUND_SOCKET_ERROR`
- `EVENT_MESSAGE_RECEIVED`
- `EVENT_NETWORK_READY`
- `EVENT_NEW_INBOUND_PEER`
- `EVENT_OUTBOUND_SOCKET_ERROR`
- `EVENT_REMOVE_PEER`
- `EVENT_REQUEST_RECEIVED`
- `EVENT_UPDATED_PEER_INFO`

Examples

Check the examples folder for a few examples to demonstrate P2P library usage with some use cases.

- This particular example will run 3 nodes that will connect to each other and will then say "hi" to each other, they will then receive a response back from their peers when they receive it.
- The find city game example will run 3 nodes that will change their city randomly and also inform the other nodes in which city they are located, if they find out that they are in the same city then they stop changing their city. The app will stop when all 3 nodes are in the same city
- Finally, the connect to Lisk networks example shows how to create a lightweight P2P client that can connect to Lisk networks such as the testnet, mainnet, and the devnet and listen to various events and request data on connection events.

Passphrase

Table of Contents

Installation

Upgrade

Mnemonic

- Example

Additional validation methods

- getPassphraseValidationErrors

@liskhq/lisk-passphrase is a package which provides tools for generating and validating mnemonic passphrases.

Installation

```
$ npm install @liskhq/lisk-passphrase
```

Upgrade

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-passphrase
```

Mnemonic

This exposes the bip39 npm package for easily generating and managing BIP39-compliant mnemonic passphrases.
Please refer to their documentation for full usage.

Example

```
import * as passphrase from '@liskhq/lisk-passphrase';

const { Mnemonic } = passphrase;

const passphrase = Mnemonic.generateMnemonic(); // 'drastic spot aerobic web wave tourist library first sco
const japanesePassphrase = Mnemonic.generateMnemonic(null, null, Mnemonic.wordlists.japanese); // 'こやく  そ
Mnemonic.validateMnemonic(japanesePassphrase, Mnemonic.wordlists.japanese); // true
```

Additional validation methods

In addition to the validation provided by the BIP39 library, a helper function is provided in order to help understand which errors may have occured with an invalid passphrase.

getPassphraseValidationErrors

This returns an array of validation errors to help with usability.

Syntax

```
getPassphraseValidationErrors(passphrase, [wordlist], expectedWords)
```

Parameters

`passphrase`: This is the candidate's passphrase to validate.

`wordlist`: The wordlist for the passphrase, (default is English). Word lists are provided by the BIP39 library described above.

`expectedWords`: The number of words of the passphrase. 12 is the default value in the case where it is undefined.

Return value

`array`: An array of errors containing details about why a passphrase is invalid. The array is empty if the passphrase is valid.

Examples

```
import * as passphrase from '@liskhq/lisk-passphrase';

const errors = passphrase.validation.getPassphraseValidationErrors('this passphrase is not    valid', Mnemo
/* [
  {
    actual: 5,
    code: 'INVALID_AMOUNT_OF_WORDS',
    expected: 24,
    message: 'Passphrase contains 5 words instead of expected 24. Please check the passphrase.'
  },
  {
    actual: 7,
    code: 'INVALID_AMOUNT_OF_WHITESPACES',
    expected: 23,
    location: [ 23, 24, 25 ],
    message: 'Passphrase contains 7 whitespaces instead of expected 23. Please check the passphrase.'
  },
  {
    actual: false,
    code: 'INVALID_MNEMONIC',
    expected: true,
    message: 'Passphrase is not a valid mnemonic passphrase. Please check the passphrase.'
  }
]*/
```

Transaction pool

Table of Contents

Installation

Update

Constants

- Usage

Terminology

Specifications

Data structures

Life cycle of transactions in the pool

Transaction pool jobs

Transaction pool actions

Transaction pool events

The @liskhq/lisk-transaction-pool package collects, verifies, and provides transactions to the forger module in order to produce a block.

Due to dynamic transaction fees and transaction nonces supported by the Lisk protocol, the transaction pool is required to sort transactions based on a nonce while also maximizing the fee.

The transaction pool has a single list of transactions and a periodic job which sorts and cleans the list as necessary.

Installation

```
npm install @liskhq/lisk-transaction-pool
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-transaction-pool
```

Constants

Transaction-pool-specific constants include names as shown below:

Usage

```
const transactionPool = require('@liskhq/lisk-transaction-pool')

transactionPool.EVENT_ADDED_TRANSACTIONS; // 'transactionsAdded'
transactionPool.EVENT_REMOVED_TRANSACTIONS; // 'transactionsRemoved'
transactionPool.EVENT_VERIFIED_TRANSACTION_ONCE; // 'transactionVerifiedOnce'

transactionPool.ACTION_ADD_VERIFIED_REMOVED_TRANSACTIONS; // 'addVerifiedRemovedTransactions'
transactionPool.ACTION_REMOVE_CONFIRMED_TRANSACTIONS; // 'removeConfirmedTransactions'
transactionPool.ACTION_ADD_TRANSACTIONS; // 'addTransactions'
transactionPool.ACTION_EXPIRE_TRANSACTIONS; // 'expireTransactions'
transactionPool.ACTION_PROCESS_VERIFIED_TRANSACTIONS; // 'processVerifiedTransactions'
transactionPool.ACTION_VALIDATE_RECEIVED_TRANSACTIONS; // 'validateReceivedTransactions'
transactionPool.ACTION_VERIFY_VALIDATED_TRANSACTIONS; // 'verifyValidatedTransactions'
transactionPool.ACTION_ADD_VERIFIED_TRANSACTIONS; // 'addVerifiedTransactions'
transactionPool.ACTION_ADD_PENDING_TRANSACTIONS; // 'addPendingTransactions'
```

Terminology

Invalid transaction

- A transaction that cannot be applied .
- A transaction that has a nonce value less than the nonce value stored in the account state.
- A transaction that does not satisfy the specific transaction requirements (example: fails the transaction validation or application logic) .

Processable transaction

- A transaction that has a correct nonce .
- A transaction that is applicable to the latest blockchain state.

Non-processable transaction
A transaction that is not applicable to the latest blockchain state, whereby the transaction nonce is larger than the account nonce .

Promote
The status update of a transaction to "processable" after applying it .

Demote
The status update of a transaction to "non-processable" due to the arrival of a new transaction, a new block or any change in the account state.

Evict
The eviction of transaction(s) from the pool when the list is full .

Discard
The removal of invalid transactions from the pool .

Reorganize
A periodic job that promotes, demotes or evicts transactions .

Specifications

- Access to an account state is required, in order to check if a transaction is non-processable or processable .
- A transaction pool has a maximum transaction limit for the whole pool (default: 4096).
- Each individual account also has its own maximum transactions limit (default: 64).
- A transaction pool can set a minimum entrance fee priority for transactions to be included.
- A transaction pool can set the minimum replacement fee difference for previous transaction invalidation and replacement.
- A user can opt to invalidate a previously sent transaction by resending a new transaction with the same nonce and a higher fee.
- Transactions in the pool expire after a set amount of time (default: 3 hours).

Data structures

transactionList
A list of all the valid transactions (both non-processable or processable) which have arrived via the transport module.

transactionAll
A list of all received and pending transactions, for checking the existence of, or returning by interface the transactions in the pool.

Life cycle of transactions in the pool

1. Unknown ⇒ non-processable .
   a. Schema must be valid .
   b. Signatures must be valid .
   c. `Transaction.nonce >= account.nonce`.
   d. All state changes must be valid.
2. Unknown ⇒ processable.
   a. All conditions from 1, except `transaction.nonce > account.nonce` .
   b. `transaction.nonce === account.nonce `.
3. No other transactions in the pool from the same address , with the same nonce.
4. Non-processable ⇒ processable (Promote).
   a. `Transaction.nonce == account.nonce || transaction.nonce == otherTransactionFromTheSameSender.nonce + 1` .
   b. All state changes per transaction are valid.
   c. All transactions from the same account which have a correct nonce are all valid when applied together .
5. Processable ⇒ non-processable .
   a. New block arrives and causes some transactions to become invalid .
   b. A replacement transaction arrives for a processable transaction with the same nonce, and hence it makes transactions which have a higher nonce unprocessable.
   c. Forger reverts a block and adds transactions back to the transaction pool, which makes transactions from the same account which have a higher nonce unprocessable.

Transaction pool jobs

Reorganize

1. Checks each account for promotable transactions .
2. Promotes all non-processable transactions which can be processable by applying them together with existing processable transactions .
3. Discards all invalid transactions and all subsequent transactions .

Evict
Evicts all transactions which exceed the expiry time and updates all subsequent transactions from the same account to be non-processable .

Transaction pool actions

removeTransaction
Removes a transaction from the list. All transactions from the same account with a larger nonce are updated to be non-processable.

addTransaction
Adds a transaction to the list. If it is a replacement transaction and has a higher fee, it replaces the older one and subsequent transactions are updated to be non-processable. If the transaction pool is full and an incoming transaction has a higher fee priority, it should accept the incoming transaction and evict another transaction according to the following rules:

- Remove non-processable transactions with the lowest feePriority.
- Remove the lowest feePriority transaction from within the highest nonce transactions for each account .
- Apply the transaction using applyFunction and check if it is PROCESSABLE, UNPROCESSABLE or INVALID.
- If PROCESSABLE or UNPROCESSABLE then add it to transactionList and feePriorityQueue, if INVALID then a relevant error is returned.

getProcessableTransactions
Provides a copy of processable transactions with a map of accounts to processable transactions, ordered by the nonce.

getAllTransactions
Provides a copy of all transactions in the pool .

getTransaction
Returns a transaction by ID for provision to another node.

Transaction pool events

- On block save: Remove transactions from the pool.
- On block delete: Adds transactions back to the pool.
- On transaction receive:
  - Checks if the transaction exists in the pool, then adds it to the pool if the nonce is not lower than the account.
  - Rejects the transaction if the nonce is lower than the account.
  - If the transaction is the same nonce:
    - Rejects the transaction if it has a lower fee.
    - Replaces the transaction if it has a higher fee.
  - If the transaction is non-sequential but it has a larger nonce, it remains in the pool for a limited time.

Transactions

Table of Contents

Installation

Upgrade

Signing transactions

- signTransaction
- signMultiSignatureTransaction
- getBytes
- getSigningBytes

Changing between formats

- convertBeddowsToLSK
- convertLSKToBeddows

Validating transactions

- validateTransaction

Calculating the minimum fee for a transaction

- computeMinFee

Transaction utility functions are defined in `@liskhq/lisk-transactions`. The functionalities can be accessed through package `@liskhq/lisk-transactions`, `@liskhq/lisk-client` or `lisk-sdk`.

Installation

```
$ npm install @liskhq/lisk-transactions
```

Upgrade

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-transactions
```

Signing transactions

signTransaction

Signs a transaction with a given networkIdentifier and passphrase. This returns a transaction object with a signature.

```
signTransaction(
  // schema defined in custom asset to be signed
  assetSchema: Schema,
  // transaction object in JavaScript object format
  transactionObject: Record<string, unknown>,
  networkIdentifier: Buffer,
  passphrase: string,
): Record<string, unknown>;
```

signMultiSignatureTransaction

Signs a transaction with a given passphrase as a multi signature account. This method can be used for signing registerMultisignatureGroup by setting the includeSenderSignature to true.

```
signMultiSignatureTransaction(
  // schema defined in the custom asset to be signed
  assetSchema: Schema,
  // transaction object in JavaScript object format
  transactionObject: Record<string, unknown>,
  networkIdentifier: Buffer,
  passphrase: string,
  // keys which correspond to the sender account of the transaction
  keys: {
    mandatoryKeys: Buffer[],
    optionalKeys: Buffer[],
  },
  // By setting true, signature corresponds to the sender public key will also be included in the signature
  // This should be used for registerMultisignatureGroup transaction
  includeSenderSignature = false,
): Record<string, unknown>
```

getBytes

Validates the transaction against the asset schema and returns transaction bytes.

```
getBytes (
  // schema defined in custom asset
  assetSchema: Schema,
  // transaction object in JavaScript object format
  transactionObject: Record<string, unknown>,
): Buffer;
```

getSigningBytes

Validates the transaction against the asset schema and returns transaction bytes for signing.

```
getSigningBytes (
  // schema defined in custom asset to be signed
  assetSchema: Schema,
  // transaction object in JavaScript object format
  transactionObject: Record<string, unknown>,
): Buffer;
```

Changing between formats

convertBeddowsToLSK

This converts amounts in Beddows (the smallest denomination), to the amounts in one LSK.

Syntax

```
convertBeddowsToLSK(amount)
```

Parameters

`amount`: `string` decimal representation of amount to be converted.

Return value

`string`: Amount in LSK.

Examples

```
transactions.convertBeddowsToLSK('100000'); // '0.001'
```

convertLSKToBeddows

This converts the amounts in LSK to the amounts in Beddows, (the smallest denomination).

Syntax

```
convertLSKToBeddows(amount)
```

Parameters

`amount`: `string` decimal representation of amount to be converted.

Return value

`string`: Amount in Beddows.

Examples

```
transactions.convertLSKToBeddows('0.001'); // '100000'
```

Validating transactions

validateTransaction

Statically validates the transaction object input using the schema.

```
validateTransaction(
  // schema defined in custom asset to be signed
  assetSchema: Schema,
  // transaction object in JavaScript object format
  transactionObject: Record<string, unknown>,
): LiskValidationError | Error | undefined;
```

Calculating the minimum fee for a transaction

computeMinFee

Returns the minimal fee for a transaction.

```
computeMinFee(
// schema defined in custom asset to be signed
  assetSchema: Schema,
  // transaction object in JavaScript object format
  transactionObject: Record<string, unknown>,
  options?: {
    minFeePerByte: number,
    baseFees: { moduleID: number, assetID: number, baseFee: number}[],
    numberOfSignatures: number,
  }
): bigint
```

Tree

Table of Contents

Installation

Update

@liskhq/lisk-tree is a package which contains Merkle tree implementations for use with Lisk-related software.

Installation

```
npm install @liskhq/lisk-tree
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-tree
```

Utils

Table of Contents

Installation

Update

Method reference

- bufferArrayContains
- bufferArrayContainsSome
- bufferArrayEqual
- bufferArrayIncludes
- bufferArrayOrderByLex
- bufferArraySubtract
- bufferArrayUniqueItems
- mergeDeep

@liskhq/lisk-utils is a library providing various utilities for handling data structures, objects, and jobs.

Installation

```
npm install @liskhq/lisk-utils
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-utils
```

Method reference

bufferArrayContains

Checks, if a Buffer array(`arr1`) contains another Buffer array(`arr2`).

Syntax

```
objects.bufferArrayContains(arr1,arr2)
```

Parameters

1. `arr1`(Buffer[]): A buffer array.
2. `arr2`(Buffer[]): Another buffer array.

Return value

`boolean`: `true` if `arr1` contains `arr2`, `false` otherwise.

bufferArrayContainsSome

Checks, if a Buffer array(`arr1`) contains parts of another Buffer array(`arr2`).

Syntax

```
objects.bufferArrayContainsSome(arr1,arr2)
```

Parameters

1. `arr1`(Buffer[]): A buffer array.
2. `arr2`(Buffer[]): Another buffer array.

Return value

`boolean`: `true` if `arr1` contains some parts of `arr2`, `false` otherwise.

bufferArrayEqual

Checks if two given Buffer arrays are identical.

Syntax

```
objects.bufferArrayEqual(arr1,arr2)
```

Parameters

1. `arr1`(Buffer[]): A buffer array.
2. `arr2`(Buffer[]): Another buffer array.

Return value

`boolean`: `true` if `arr1` is equal to `arr2`, `false` otherwise.

bufferArrayIncludes

Checks, if a Buffer array includes a certain value.

Syntax

```
objects.bufferArrayIncludes(arr,val)
```

Parameters

1. `arr`(Buffer[]): A buffer array.
2. `val`(Buffer): A buffer value.

Return value

`boolean`: `true` if `arr` includes `val`, `false` otherwise.

bufferArrayOrderByLex

Checks if a given buffer array is sorted in lexicographical order.

Syntax

```
objects.bufferArrayOrderByLex(arr1)
```

Parameters

1. `arr1`(Buffer[]): A buffer array.

Return value

`boolean`: `true` if `arr1` is sorted in lexicographical order, `false` otherwise.

bufferArraySubtract

Substracts one buffer array from another buffer array.

Syntax

```
objects.bufferArraySubtract(arr1,arr2)
```

Parameters

1. `arr1`(Buffer[]): A buffer array.
2. `arr2`(Buffer[]): Another buffer array.

Return value

`Buffer[]`: A new buffer array, which contains the values of `arr1` excluding the values of `arr2`, if present in `arr1`.

bufferArrayUniqueItems

Checks if all items of a buffer array are unique.

Syntax

```
objects.bufferArrayUniqueItems(arr1)
```

Parameters

1. `arr1`(Buffer[]): A buffer array.

Return value

`boolean`: `true` if all items in the buffer array are unique, `false` otherwise.

mergeDeep

Merges objects into one object.

Removes properties which are `null` or `undefined` during the merge.

Syntax

```
objects.mergeDeep(dest, obj1, obj2, ...)
```

Parameters

1. `dest`(object): The other objects will be merged into this one.
2. `obj1`(object): An object.
3. `obj2`(object): An object.

Return value

`object`: The merged object.

Validator

Table of Contents

- Installation
- Update
- Usage
- Constants
- Validation methods
  - isBoolean
  - isBytes
  - isCsv
  - isEncryptedPassphrase
  - isHexString
  - isIP
  - isIPV4
  - isIPV6
  - isNumberString
  - isPort
  - isProtocolString
  - isRangedSemVer
  - isSint32
  - isSint64
  - isSemVer
  - isString
  - isStringEndsWith
  - isUint32
  - isUint64
  - isValidInteger
  - isLessThanRangedVersion
  - isGreaterThanRangedVersion
  - isVersionMatch

The @liskhq/lisk-validator package is a wrapper of validation packages. It includes custom validations utilities related to the Lisk Protocol.

Installation

```
npm install @liskhq/lisk-validator
```

Update

To perform an upgrade, execute the following command:

```
npm update @liskhq/lisk-validator
```

Usage

```
const validator = require('@liskhq/lisk-validator');

console.log(validator.MAX_SINT64); //9223372036854775807n
```

Constants

```
MAX_SINT32 : 2147483647,
MIN_SINT32 : -2147483647,
MAX_SINT64 : 9223372036854775807n,
MIN_SINT64 : -9223372036854775807n,
MAX_UINT64 : 9223372036854775807n,
MIN_UINT64 : -9223372036854775808n
```

Validation methods

isBoolean

Checks if a given input is of type `boolean`.

Syntax

```
isBoolean(data);
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type boolean, `false` otherwise.

isBytes

Checks if a given input is a Buffer.

Syntax

```
isBytes(data);
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type Buffer, `false` otherwise.

isCsv

Checks, if a given input is data in CSV format.

Syntax

```
isCsv(data)
```

Parameters

`data`(string): String to be checked.

Return value

`boolean` `true` if the provided input is in CSV format, `false` otherwise.

isEncryptedPassphrase

Checks if a given input is an encrypted passphrase.

Syntax

```
isEncryptedPassphrase(data)
```

Parameters

`data`(string): String to be checked.

Return value

`boolean` `true` if the provided data is an encrypted passphrase, `false` otherwise.

isHexString

Checks if a given input is a hex string.

Syntax

```
isHexString(data);
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is a hex string, `false` otherwise.

isIP

Checks if a given string is an IP.

Syntax

```
isIP(data);
```

Parameters

`data`(string): String to be checked.

Return value

`boolean` `true` if the provided data is an IP, `false` otherwise.

isIPV4

Checks if a given string is an IPv4 (Internet Protocol Version 4).

Syntax

```
isIPV4(data);
```

Parameters

`data`(string): String to be checked.

Return value

`boolean` `true` if the provided data is an IPv4, `false` otherwise.

isIPV6

Checks if a given string is an IPv6 (Internet Protocol Version 6).

Syntax

```
isIPV6(data);
```

Parameters

`data`(string): String to be checked.

Return value

`boolean` `true` if the provided data is an IPv6, `false` otherwise.

isNumberString

Checks if a given input is a number string.

Syntax

```
isNumberString(num)
```

Parameters

`num`- Data to be checked.

Return value

`boolean` `true` if the provided data is a number string, `false` otherwise.

isPort

Checks if a given string is a port.

Syntax

```
isPort(port);
```

Parameters

`port`(string): String to be checked.

Return value

`boolean` `true` if the provided data is a port, `false` otherwise.

isProtocolString

Checks if a given string is a protocol.

Syntax

```
isProtocolString(data)
```

Parameters

`data`(string): String to be checked.

Return value

`boolean` `true` if the provided data is a protocol string, `false` otherwise.

isRangedSemVer

Checks if a given input is a ranged lil SemVer (Semantic Versioning), version.

Syntax

```
isRangedSemVer(version)
```

Parameters

`version`(string): String to be checked.

Return value

`boolean` `true` if the provided data is a valid SemVer version string, `false` otherwise.

isSint32

Checks if a given input is of type `SInt32`.

Syntax

```
isSint32(data)
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type sint32, `false` otherwise.

isSint64

Checks if a given input is of type `SInt64`.

Syntax

```
isSint64(data)
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type sint64, `false` otherwise.

isSemVer

Checks if a given input is a SemVer version.

Syntax

```
isSemVer(version)
```

Parameters

`version`(string): String to be checked.

Return value

`boolean` `true` if the provided data is a SemVer version, `false` otherwise.

isString

Checks if a given input is of type `string`.

Syntax

```
isString(data)
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type `string`, `false` otherwise.

isStringEndsWith

Checks if a string ends with a given list of suffixes.

Syntax

```
isStringEndsWith(target,suffixes)
```

Parameters

1. `target`(string): String to be checked.
2. `suffixes`(array[]): Suffixes to be checked.

Return value

`boolean` `true` if the provided string ends with one of the suffixes, `false` otherwise.

isUint32

Checks if a given input is of type `UInt32`.

Syntax

```
isUint32(data)
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type uint32, `false` otherwise.

isUint64

Checks if a given input is of type `UInt64`.

Syntax

```
isUint64(data)
```

Parameters

`data`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type uint64, `false` otherwise.

isValidInteger

Checks if a given input is a valid integer.

Syntax

```
isValidInteger(num)
```

Parameters

`num`: Data to be checked.

Return value

`boolean` `true` if the provided data is of type integer, `false` otherwise.

isLessThanRangedVersion

isGreaterThanRangedVersion

isVersionMatch