Welcome to REI Network

REI Network is not a new project, but an upgraded version of GXChain. REI was developed to better adapt to the development trend of the blockchain, and to achieve a lightweight, Ethereum compatible, higher performance, and no fee blockchain framework

New Features

- Compatible with **EVM**
- Compatible with **Ethereum's RPC** and **Websocket** interfaces, and can support **GRPC** in the future
- Rewrite the network module and use **Libp2p** instead of **Devp2p** (current Ethereum client solution), because we believe that Libp2p has better standards and can achieve better versatility and scalability
- Realize lower resource consumption through the design of tokenomics (**Gas Free**)
- Realize **Systems contracts** that can be soft-forked and upgraded. These contracts include: Staking/Slashing, ResourceManager and IBC contracts, etc.
- Achieve a more efficient and more random consensus: **DPoS+BFT**, to ensure decentralization and be more green power
- Realize the **abstract consensus module**, so that the code of REI Network can be easily combined and become a chain-making tool

Entry Points

REI Mainnet Settings:

- **Network Name**: REI Network
- **RPC URL**: https://rpc.rei.network
- **ChainID**: `47805`
- **Symbol**: `REI`
- **Decimals**: `18`
- **Explorer**: https://scan.rei.network

REI Testnet Settings:

- **Network Name**: REI Testnet
- **RPC URL**: https://rpc-testnet.rei.network
- **ChainID**: `12357`
- **Symbol**: `REI`
- **Decimals**: `18`
- **Explorer**: https://scan-test.rei.network

Quick Start

> A guide to start REI Network Node

Quick Start

API Reference

> Introduction to REI Network JSON-RPC specifications

API Reference

System Contracts

> Introduction to REI Network System contracts

System Contracts

> **Good to know:** The document is constantly being updated and improved, and you are welcome to ask us questions and suggestions

Guides

Exchange Integration

Using The Graph

Using Gnosis Safe

Token Addresses

Why We build REI Network

When we had the first idea about blockchain in the year 2016, we wanted to develop a high-performance blockchain that could adapt to the growing business demands, so we adopted the Graphene framework to build the initial version of GXChain 1.0. With the continuous iterative update of the technology, we have introduced the WASM virtual machine on Graphene, which supports mainstream programming languages to write smart contracts, and implements one token - one voting power, and an innovative staking&voting mechanism on the DPoS election mechanism. GXChain 1.0 has completed its historical mission within a period of time and has won support from numerous community users. At the end of 2019, when we witnessed DeFi booming, we considered whether GXChain could provide a more friendly experience for these DeFi projects. When we tried to implement these basic ideas of DeFi on GXChain 1.0, we found some problems:

- A lot of DeFi projects are built on Ethereum/EVM, and most of them adopt Solidity programming language
- Although Ethereum is not very user-friendly (low efficiency and high cost), it seems developers and users care little about it.

So we consider that what really attracts these developers. After a period of research, we found:

- EVM has a more mature and historically tested instruction set, as well as a friendly compiler & editor, and other infrastructure, and has accumulated a large number of developers on these foundations
- The ecology of EVM has better standards, such as ERC20, ERC721, etc., and the combination of these standards is very strong
- The Ethereum community has accumulated a large number of high-quality projects and loyal users in the early days. Although Ethereum is not very user-friendly, it is still the first choice for DeFi projects.

Therefore, we decided to make GXChain 2.0 compatible with EVM and its ecological infrastructure, so that DeFi developers and their applications can seamlessly migrate to GXChain 2.0, maintaining the same experience as the Ethereum blockchain, and improving consensus efficiency (the performance of GXChain2.0), to reduce the cost of use. After having such preliminary ideas, we quickly refined some specific technical solutions, that is, the technical characteristics of GXChain 2.0:

- Compatible with **EVM**
- Compatible with **Ethereum's RPC** and **Websocket** interfaces, and can support **GRPC** in the future
- Rewrite the network module and use **Libp2p** instead of **Devp2p** (current Ethereum client solution), because we believe that Libp2p has better standards and can achieve better versatility and scalability
- Realize lower resource consumption through the design of tokenomics (**Gas Free**)
- Realize **Systems contracts** that can be soft-forked and upgraded. These contracts include: Staking/Slashing, ResourceManager and IBC contracts, etc.
- Achieve a more efficient and more random consensus: **DPoS+BFT**, to ensure decentralization and be more green power
- Realize the **abstract consensus module**, so that the code of REI Network can be easily combined and become a chain-making tool

In the face of the above-mentioned technical ideals, we hope that the code of GXChain 2.0 will no longer be a simple Fork or depending on an open-sourced chain-making framework, but on the basis of EVM, reconstruct the network, consensus, node election, and RPC modules. We hope these codes are more concise and modular so that they have better composability.

Introduction to $REI

What is $REI?

`$REI` is the native token of REI Network which secures the running of REI Network. `$REI` has the following usages:

- **Gas Fee**: When you send transaction on the REI Network , you can pay a small gas fee using `$REI` , This fee is a reward to the node for processing and verifying the transaction.
- **Governance Token**: `$REI` can be used for REI DAO management, `$REI` holders can commit proposals and vote for system level proposals on https://dao.rei.network
- **Validator Pledge Token**: `$REI` can be used as a pledge token for node who want's to become a validator of REI Network
- **Voting token**: `$REI` is the only voting token for node election
- **Stake for Free Gas**: `$REI` holders are able to stake `$REI` to own `$Crude` , which can be used instead of gas

What is the responsibility of the $REI holder?

- The responsibility of REI holders is primarily to ensure the stable and secure operation and ecological growth of the REI Network public chain by building governance processes and infrastructure that work, and to defend against proposals that do not support the core objectives of governance, which is in the core interest of REI holders.
- REI holders need to openly communicate and express their views on the various issues being addressed by REI governance at any given time and seriously participate in the forum discussions; the content of the proposal and the logic behind it is one of the main criteria for measuring the quality of the proposal.

Quick Start

A guide to start REI Network Node

Install

Install from npm:

```
npm install @rei-network/cli --global
```

Install from source code:

```
git clone https://github.com/rei-network/rei.git
cd rei
npm install
npm link -w @rei-network/cli
```

Usage

```
Usage: rei [options] [command]

Options:
  -V, --version                             output the version number
  --rpc                                     open rpc server
  --rpc-port <port>                         rpc server port
  --rpc-host <port>                         rpc server host
  --rpc-api <apis>                          rpc server apis: debug, eth, net, txpool, w
  --disable-p2p                             disable p2p server
  --p2p-tcp-port <port>                     p2p server tcp port
  --p2p-udp-port <port>                     p2p server udp port
  --p2p-nat <ip>                            p2p server nat ip
  --max-peers <peers>                       max p2p peers count
  --max-dials <dials>                       max p2p dials count
  --bootnodes <bootnodes...>                comma separated list of bootnodes
  --datadir <path>                          chain data dir path (default: "~/.rei")
  --keystore <keystore>                     the datadir for keystore (default: "keystor
  --unlock <unlock>                         comma separated list of accounts to unlock
  --password <password>                     password file to use for non-interactive pa
  --chain <chain>                           chain name: rei-mainnet, rei-testnet, rei-d
  --mine                                    mine block
  --coinbase <address>                      miner address
  --verbosity <verbosity>                   logging verbosity: silent, error, warn, inf
  --receipts-cache-size <receiptsCacheSize> receipts cache size
  -h, --help                                display help for command

Commands:
  account                                   Manage accounts
```

Default settings:

Params	Default Value	Description
P2P TCP Port	4191	P2P Network TCP Port
P2P UDP Port	9810	P2P Network UDP Port
P2P NAT	127.0.0.1	The public IP of this node, used for node discovery
RPC Port	11451	
RPC host	127.0.0.1	
RPC API	eth,net,web3	

Example

Proposer node startup

```
rei --mine --coinbase 0x...abc --unlock 0x...abc --password ./password
```

RPC node startup

```
rei --rpc --rpc-host 0.0.0.0
```

Bootnode startup

```
rei --p2p-nat 1.2.3.4
```

Testnet or devnet node startup

```
rei --chain rei-testnet
rei --chain rei-devnet
```

API Reference

Introduction to REI Network JSON-RPC specifications

REI Network's RPC interfaces are fully compatible with Ethereum (except EIP1559), so you can refer to ETH's JSON-RPC specifications

OPEN-RPC Playground

Ethereum JSON-RPC Specification

Good to know: REI Network's RPC module currently implements all APIs of `eth_namespace`, `debug_namespace` and `txpool_namespace`; if you have other namespace API requirements, please feel free to give feedback to the team or submit a Pull Request

Token Addresses

Name	Symbol	Decimals	Testnet	Mainnet
Wrapped REI	WREI	18	0x2545AF3D8b11e295bB7aEdD5826021AB54F71630	0x2545AF3 5bB7aEdD5 54F71630
Wrapped BTC	WBTC	8	0xa701aFffa73F6Ddac3b30a12dae9Ff30Ef692f58	0x8059E67 db5155bF4 fDc5561
Tether USD	USDT	6	0xfAA4C7dB13c28B693DBbC874bd0CD97E27737CFE	0x988a631 Bb77EE0f5 5dcfceC7
USD Coin	USDC	6	0x7eDe327Df76BF39b40FD1FD73C294B88568db221	0x8d5E122 2E09A3AB 486f53314
Wrapped Ether	WETH	18	0x2EBA157cBF9CbC4AD2631AA0c4F682C2fC6e0dC1	0x7a53134 Afb2Cf5ec 0fc2884A
Dai Stablecoin	DAI	18	0xE585f7FD1339781A196194185160A2DF5716A741	0x0ba8598 D77fBb494 6E4FB1f6

System Contracts

Introduction to REI Network System Contracts

Public Modules

Contract Name	Description	Address
StakeManager	A smart contract used for node election	`0x0000000000000000000000000000000000000000000000001001`
Fee	A smart contract that accepts native token(REI) deposit and calculates user fees	`0x0000000000000000000000000000000000000000000000001005`
ContractFee	A smart contract for registering contract creators and setting contract fee	`0x000000000000000000000000000000000000000000000000100b`

Private Modules

Contract Name	Description	Address
Config	Global config contract	`0x0000000000000000000000000000000000000000000000001000`
UnstakePool	Un-stake asset pool	`0x0000000000000000000000000000000000000000000000001001`
ValidatorRewardPool	A smart contract that keeps validator reward for validator	`0x0000000000000000000000000000000000000000000000001004`
CommmissionShare	A smart contract that keeps commission reward for all staking user, dynamically deployed for each validator	-
FreeFee	A smart contract that calculates user daily free fees	`0x0000000000000000000000000000000000000000000000001006`
FeePool	A smart contract that assigns REI rewards to miners according to miner shares every 24 hours	`0x0000000000000000000000000000000000000000000000001007`
Router	A router smart contract, the blockchain system will only interact with router contract	`0x0000000000000000000000000000000000000000000000001008`
FeeToken	An ERC20 smart contract, only provides `balanceOf` method for users to query the fee(Stake to own) balance	`0x0000000000000000000000000000000000000000000000001009`
FreeFeeToken	An ERC20 smart contract, only provides `balanceOf` method for users to query the free fee(Free to own) balance	`0x000000000000000000000000000000000000000000000000100a`

> The blockchain system will only send transaction to **Router** contract

StakeManager

ON THIS PAGE
Contents
Globals
Modifiers
onlyRouterOrFeePool
Functions
constructor
indexedValidatorsLength
indexedValidatorsExists
indexedValidatorsByIndex
indexedValidatorsById
getVotingPowerByIndex
getVotingPowerById
getVotingPowerByAddress
getTotalLockedAmountAndValidat...
activeValidatorsLength
estimateSharesToAmount
estimateAmountToShares
estimateUnstakeAmount
receive
stake
startUnstake
unstake
removeIndexedValidator
addIndexedValidator
reward
slash
onAfterBlock
Events
Reward

Contents

Globals

Var	Type
validatorId	uint256
validators	mapping(address => struct Validator)
unstakeId	uint256
unstakeQueue	mapping(uint256 => struct Unstake)
totalLockedAmount	uint256
activeValidators	struct ActiveValidator[]
proposer	address

Modifiers

onlyRouterOrFeePool

No description

Declaration:

```
modifier onlyRouterOrFeePool
```

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

indexedValidatorsLength

Get the indexed validators length.

Declaration:

```
function indexedValidatorsLength(
) external returns (uint256)
```

Modifiers: No modifiers

indexedValidatorsExists

Determine whether the index validator exists by id.

Declaration:

```
function indexedValidatorsExists(
  uint256 id
) external returns (bool)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
id	uint256	The validator id

indexedValidatorsByIndex

Get indexed validator address by index.

Declaration:

```
function indexedValidatorsByIndex(
  uint256 index
) external returns (address validator)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
index	uint256	The validator index

indexedValidatorsById

Get indexed validator address by id.

Declaration:

```
function indexedValidatorsById(
  uint256 id
) external returns (address)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
id	uint256	The validator id

getVotingPowerByIndex

Get the voting power by validator index, if index is out of range or validator doesn't exist, return 0.

Declaration:

```
function getVotingPowerByIndex(
  uint256 index
) external returns (uint256)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
index	uint256	The validator index

getVotingPowerById

Get the voting power by validator id, if doesn't exist, return 0.

Declaration:

```
function getVotingPowerById(
  uint256 id
) external returns (uint256)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
id	uint256	The validator id

getVotingPowerByAddress

Get the voting power by validator address, if the validator doesn't exist, return 0.

Declaration:

```
function getVotingPowerByAddress(
  address validator
) public returns (uint256)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
validator	address	Validator address

getTotalLockedAmountAndValidatorCount

Get the total locked amount and the validator count, but no including the `excludes`.

Declaration:

```
function getTotalLockedAmountAndValidatorCount(
  address[] excludes
) external returns (uint256 _totalLockedAmount, uint256 validatorCount)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
excludes	address[]	Excluded addresses

activeValidatorsLength

Get the active validators list length.

Declaration:

```
function activeValidatorsLength(
) external returns (uint256)
```

Modifiers: No modifiers

estimateSharesToAmount

Estimate how much GXC should be stake, if user wants to get the number of shares, Or estimate how much GXC can be obtained, if user unstake the amount of GXC.

Declaration:

```
function estimateSharesToAmount(
  address validator,
  uint256 shares
) public returns (uint256 amount)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
validator	address	Validator address
shares	uint256	Number of shares

estimateAmountToShares

Estimate how much shares should be unstake, if user wants to get the amount of GXC. Or estimate how much shares can be obtained, if user stake the amount of GXC. If the validator doesn't exist, return 0.

Declaration:

```
function estimateAmountToShares(
  address validator,
  uint256 amount
) public returns (uint256 shares)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
validator	address	Validator address
amount	uint256	Number of GXC

estimateUnstakeAmount

Estimate how much GXC can be claim, if unstake the number of shares(when unstake timeout). If the validator doesn't exist, return 0.

Declaration:

```
function estimateUnstakeAmount(
  address validator,
  uint256 shares
) external returns (uint256 amount)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
validator	address	Validator address
shares	uint256	Number of shares

receive

No description

Declaration:

```
function receive(
) external
```

Modifiers: No modifiers

stake

Stake for validator and mint share token to `to` address. It will emit `Stake` event.

Declaration:

```
function stake(
  address validator,
  address to
) external nonReentrant returns (uint256 shares)
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
validator	address	Validator address
to	address	Receiver address

startUnstake

Start unstake shares for validator. Stake manager will burn the shares immediately, but return GXC to `to` address after `config.unstakeDelay`. It will emit `StartUnstake` event.

Declaration:

```
function startUnstake(
  address validator,
  address payable to,
  uint256 shares
) external nonReentrant returns (uint256)
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
validator	address	Validator address
to	address payable	Receiver address
shares	uint256	Number of shares to be burned

Returns:

Type	Description
Unstake	id(if amount is zero, return MAX_UINT256)
### startClaim	
Start claim validator reward.	
Stake manager will claim GXC from validator reward manager immediately, but return GXC to `to` address after `config.unstakeDelay`.	
It will emit `StartUnstake` event.	

Declaration:

```
function startClaim(
  address payable to,
  uint256 amount
) external nonReentrant returns (uint256)
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
to	address payable	Receiver address
amount	uint256	Number of GXC

Returns:

Type	Description
Unstake	id
### setCommissionRate	
Set validator commission rate.	

Declaration:

```
function setCommissionRate(
  uint256 rate
) external nonReentrant
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
rate	uint256	New commission rate

unstake

Unstake by id, return unstake amount.

Declaration:

```
function unstake(
  uint256 id
) external nonReentrant returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
id	uint256	Unstake id

removeIndexedValidator

Remove the validator from `indexedValidators` if the voting power is less than `minIndexVotingPower` This can be called by anyone.

Declaration:

```
function removeIndexedValidator(
  address validator
) external nonReentrant
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
validator	address	Validator address

addIndexedValidator

Add the validator to `indexedValidators` if the voting power is greater than `minIndexVotingPower` This can be called by anyone.

Declaration:

```
function addIndexedValidator(
  address validator
) external nonReentrant
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
validator	address	Validator address

reward

Reward validator, only can be called by system caller

Declaration:

```
function reward(
  address validator
) external nonReentrant onlyRouterOrFeePool
```

Modifiers:

Modifier
nonReentrant
onlyRouterOrFeePool

Args:

Arg	Type	Description
validator	address	Validator address

slash

Slash validator, only can be called by system caller

Declaration:

```
function slash(
  address validator,
  uint8 reason
) external nonReentrant onlyRouter returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant
onlyRouter

Args:

Arg	Type	Description
validator	address	Validator address
reason	uint8	Slash reason

onAfterBlock

After block callback, it will be called by system caller after each block is processed

Declaration:

```
function onAfterBlock(
  address _proposer,
  address[] acValidators,
  int256[] priorities
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

Args:

Arg	Type	Description
_proposer	address	Proposer address
acValidators	address[]	Active validators list
priorities	int256[]	Priority list of active validators

Events

Reward

Emitted when a validator gets a reward NOTE: this event is never shown in the block, because the `reward` function is only called by the system caller

Params:

Param	Type	Indexed	Description
validator	address	✅	Validator address
value	uint256	✅	Reward amount
### Slash			
Emitted when a validator is slashed			
NOTE: this event is never shown in the block,			

```
because the `slash` function is only called by the system caller
```

Params:

Param	Type	Indexed	Description
validator	address	✅	Validator address
value	uint256	✅	Slashed amount
### Stake			
Emitted when the user stakes			

Params:

Param	Type	Indexed	Description
validator	address	✅	Validator address
value	uint256	✅	Stake value
to	address		Receiver address
shares	uint256		Number of shares minted
### StartUnstake			
Emitted when the user starts unstake			

Params:

Param	Type	Indexed	Description
id	uint256	✅	Unique unstake id
validator	address	✅	Validator address
value	uint256	✅	Stake value
to	address		Receiver address
unstakeShares	uint256		Number of unstake shares to be burned
timestamp	uint256		Release timestamp
### DoUnstake			
Emitted when stake manager `unstake`			

Params:

Param	Type	Indexed	Description
id	uint256	✅	Unique unstake id
validator	address	✅	Validator address
to	address		Receiver address
amount	uint256		GXC Released
### SetCommissionRate			
Emitted when validator set commission rate			

Params:

Param	Type	Indexed	Description
validator	address	✅	Validator address
rate	uint256	✅	New commission
timestamp	uint256	✅	Update timestamp
### IndexedValidator			
Emitted when a new validator is indexed			

Params:

Param	Type	Indexed	Description
validator	address	✅	Validator address
votingPower	uint256	✅	Validator voting power
### UnindexedValidator			
Emitted when a new validator is unindexed			

Params:

Param	Type	Indexed	Description
validator	address	✅	Validator address

Fee

Contents

Globals

Var	Type
userTotalAmount	mapping(address => uint256)
userUsage	mapping(address => struct IFee.UsageInfo)
userDeposit	mapping(address => mapping(address => struct IFee.DepositInfo))
totalAmount	uint256

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

deposit

Deposit amount to target user.

Declaration:

```
function deposit(
  address user
) external nonReentrant
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
`user`	address	Target user address

withdraw

Withdraw amount from target user.

Declaration:

```
function withdraw(
  address user,
  uint256 desiredAmount,
  uint256 minAmount
) external nonReentrant
```

Modifiers:

Modifier
nonReentrant

Args:

Arg	Type	Description
`user`	address	Target user address
`desiredAmount`	uint256	Desired withdraw amount
`minAmount`	uint256	Min withdraw amount

estimateWithdrawableTimestamp

Estimate wtihdrawable timestamp, if the estimation fails, return 0.

Declaration:

```
function estimateWithdrawableTimestamp(
  address user,
  address from
) external returns (uint256 timestamp)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`user`	address	Target user address
`from`	address	From user address

estimateWithdrawableAmount

Estimate wtihdrawable amount.

Declaration:

```
function estimateWithdrawableAmount(
  address user,
  uint256 timestamp
) public returns (uint256)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`user`	address	Target user address
`timestamp`	uint256	Current timestamp

estimateFee

Estimate user fee.

```
userFee = userTotalAmount * dailyFee / totalAmount - userUsage
```

Declaration:

```
function estimateFee(
  address user,
  uint256 timestamp
) public returns (uint256 fee)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`user`	address	User address
`timestamp`	uint256	Current timestamp

estimateUsage

Estimate user usage

```
T: current timestamp
T': last timestamp
userUsage': last fee usage

if T - T' < feeRecoverInterval
    userUsage = (1 - (T - T') / feeRecoverInterval) * userUsage'
else
    userUsage = 0
```

Declaration:

```
function estimateUsage(
  struct IFee.UsageInfo ui,
  uint256 timestamp
) public returns (uint256 usage)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`ui`	struct IFee.UsageInfo	Usage information
`timestamp`	uint256	Current timestamp

consume

Consume user fee, can only be called by the system caller.

Declaration:

```
function consume(
  address user,
  uint256 usage
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

Args:

Arg	Type	Description
`user`	address	User address
`usage`	uint256	Number of usage fee

Events

Deposit

Emit when user deposits.

Params:

Param	Type	Indexed	Description
`by`	address	✅	Deposit user
`to`	address	✅	Receiver user
`amount`	uint256	✅	Deposit amount
### Withdraw			
Emit when user withdraws.			

Params:

Param	Type	Indexed	Description
`by`	address	✅	Withdraw user
`from`	address	✅	From user
`amount`	uint256	✅	Withdraw amount

ContractFee

Contents

Globals

Var	Type
feeOf	mapping(address => uint256)
creatorOf	mapping(address => address)

Functions

generateAddress

Generate contract address.

Declaration:

```
function generateAddress(
  address from,
  uint256 nonce
) public returns (address)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`from`	address	Creator address
`nonce`	uint256	Creator nonce

generateAddress2

Generate contract address(create2).

Declaration:

```
function generateAddress2(
  address from,
  bytes32 salt,
  bytes32 codeHash
) public returns (address)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`from`	address	Creator address
`salt`	bytes32	Salt
`codeHash`	bytes32	Deploy code hash, notice: this is not the same as the account code hash

register

Register the contract creator.

Declaration:

```
function register(
  address parent,
  bool[] flags,
  uint256[] nonces,
  struct Create2Info[] infos
) external
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`parent`	address	Root creator address
`flags`	bool[]	A list of flags, if the flag is true, it means `create` and load a nonce from nonces, otherwise it means `create2` and load a `Create2Info` from infos
`nonces`	uint256[]	A list of nonces
`infos`	struct Create2Info[]	A list of `Create2Info`

setFee

Set contract fee. The contract fee can be set only when the sender is the creator of the contract.

Declaration:

```
function setFee(
  address contractAddress,
  uint256 fee
) external
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`contractAddress`	address	Target contract address
`fee`	uint256	Contract fee

Events

Register

Registration event, emit when someone registers their contract.

Params:

Param	Type	Indexed	Description
`parent`	address		Creator
`child`	address		Contract address

Router

Contents

Modifiers

onlySystemCaller

No description

Declaration:

```
modifier onlySystemCaller
```

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

estimateTotalFee

Estimate daily fee and free fee left.

Declaration:

```
function estimateTotalFee(
  address from,
  address to,
  uint256 timestamp
) external returns (uint256 fee, uint256 freeFee, uint256 contractFee)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`from`	address	Transaction sender
`to`	address	Transaction receiver(if contract creation, address(0))
`timestamp`	uint256	Timestamp

assignTransactionReward

Assign transaction reward to miner, and emit the `UsageInfo` event, if the consumed fee is `dailyFee` or `dailyFreeFee`, it will only increase miner's share of the fee pool, otherwise, if the consumed fee is user's balance, it will add the fee to the fee pool and increase miner's share of the fee pool.

Declaration:

```
function assignTransactionReward(
  address validator,
  address from,
  address to,
  uint256 feeUsage,
  uint256 freeFeeUsage,
  uint256 contractFeeUsage
) external nonReentrant onlySystemCaller
```

Modifiers:

Modifier
nonReentrant
onlySystemCaller

Args:

Arg	Type	Description
`validator`	address	Block miner
`from`	address	Transaction sender
`to`	address	Transaction receiver(if contract creation, address(0))
`feeUsage`	uint256	`dailyFee` usage
`freeFeeUsage`	uint256	`dailyFreeFee` usage
`contractFeeUsage`	uint256	Contract fee usage

assignBlockReward

Assign block reward, and call `onAssignBlockReward` callback, it will split the block reward into two parts according to the `minerRewardFactor`, one part will be directly distributed to miners as a reward, and the other part will be added to the transaction fee pool.

Declaration:

```
function assignBlockReward(
  address validator
) external nonReentrant onlySystemCaller
```

Modifiers:

Modifier
nonReentrant
onlySystemCaller

Args:

Arg	Type	Description
`validator`	address	Block miner

slash

Slash validator by reason

Declaration:

```
function slash(
  address validator,
  uint8 reason
) external nonReentrant onlySystemCaller
```

Modifiers:

Modifier
nonReentrant
onlySystemCaller

Args:

Arg	Type	Description
`validator`	address	Validator address
`reason`	uint8	Slash reason

onAfterBlock

After block callback, it only can be called by system caller

Declaration:

```
function onAfterBlock(
  address _proposer,
  address[] acValidators,
  int256[] priorities
) external nonReentrant onlySystemCaller
```

Modifiers:

Modifier
nonReentrant
onlySystemCaller

Args:

Arg	Type	Description
`_proposer`	address	Proposer address
`acValidators`	address[]	Parameter of StakeManager.onAfterBlock
`priorities`	int256[]	Parameter of StakeManager.onAfterBlock

Events

UsageInfo

`UsageInfo` event contains the usage information of tx, it will be automatically appended to the end of the transaction log.

Params:

Param	Type	Indexed	Description
`feeUsage`	uint256		`dailyFee` usage
`freeFeeUsage`	uint256		`dailyFreeFee` usage
`contractFeeUsage`	uint256		Contract fee usag
`balanceUsage`	uint256		Transaction send balance usage

AbstractToken

> Implementation of the {IERC20} interface.

Contents

Globals

Var	Type
name	string
symbol	string
decimals	uint8
config	contract IConfig

Functions

constructor

No description

Declaration:

```
function constructor(
) public
```

Modifiers: No modifiers

totalSupply

No description

Declaration:

```
function totalSupply(
) public returns (uint256)
```

Modifiers: No modifiers

balanceOf

No description

Declaration:

```
function balanceOf(
) public returns (uint256)
```

Modifiers: No modifiers

transfer

No description

Declaration:

```
function transfer(
) public returns (bool)
```

Modifiers: No modifiers

allowance

No description

Declaration:

```
function allowance(
) public returns (uint256)
```

Modifiers: No modifiers

approve

No description

Declaration:

```
function approve(
) public returns (bool)
```

Modifiers: No modifiers

transferFrom

No description

Declaration:

```
function transferFrom(
) public returns (bool)
```

Modifiers: No modifiers

CommissionShare

Contents

Globals

Var	Type
validator	address

Functions

constructor

No description

Declaration:

```
function constructor(
) public ERC20 Only
```

Modifiers:

Modifier
ERC20
Only

estimateSharesToAmount

Estimate how much REI should be stake, if user wants to get the number of shares, or estimate how much REI can be obtained, if user unstake the amount of REI.

Declaration:

```
function estimateSharesToAmount(
  uint256 shares
) external returns (uint256 amount)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`shares`	uint256	Number of shares

estimateAmountToShares

Estimate how much shares should be unstake, if user wants to get the amount of REI, or estimate how much shares can be obtained, if user stake the amount of REI.

Declaration:

```
function estimateAmountToShares(
  uint256 amount
) external returns (uint256 shares)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`amount`	uint256	Number of REI

mint

Mint share token to `to` address. Can only be called by stake manager.

Declaration:

```
function mint(
  address to
) external nonReentrant onlyStakeManager returns (uint256 shares)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`to`	address	Receiver address

burn

Burn shares and return REI to `to` address. Can only be called by stake manager.

Declaration:

```
function burn(
  uint256 shares
) external nonReentrant onlyStakeManager returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`shares`	uint256	Number of shares to be burned

reward

Reward validator.

Declaration:

```
function reward(
) external nonReentrant onlyStakeManager
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

slash

Slash validator and transfer the slashed amount to `address(0)`.

Declaration:

```
function slash(
  uint8 factor
) external nonReentrant onlyStakeManager returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`factor`	uint8	Slash factor.

FeeToken

Contents

Functions

constructor

No description

Declaration:

```
function constructor(
) public AbstractToken
```

Modifiers:

Modifier
AbstractToken

balanceOf

No description

Declaration:

```
function balanceOf(
) public returns (uint256)
```

Modifiers: No modifiers

FeePool

Contents

Globals

Var	Type
sharesOf	mapping(address => uint256)
totalShares	uint256
accTxFee	uint256
globalTimestamp	uint256
validators	address[]

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

validatorsLength

Get validators length.

Declaration:

```
function validatorsLength(
) external returns (uint256)
```

Modifiers: No modifiers

earn

Increase miner's share.

Declaration:

```
function earn(
  address validator,
  uint256 earned
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

Args:

Arg	Type	Description
`validator`	address	Miner address
`earned`	uint256	Miner earned share.

accumulate

Add reward to fee pool.

Declaration:

```
function accumulate(
  bool isTxFee
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

Args:

Arg	Type	Description
`isTxFee`	bool	Is transaction fee

onAssignBlockReward

Assign block reward callback, it only can be called by router.

Declaration:

```
function onAssignBlockReward(
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

ValidatorRewardPool

Contents

Globals

Var	Type
balanceOf	mapping(address => uint256)

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

claim

Claim validator reward.

Declaration:

```
function claim(
  address validator,
  uint256 amount
) external nonReentrant onlyStakeManager
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`validator`	address	Validator address.
`amount`	uint256	Claim amount.

reward

Reward validator.

Declaration:

```
function reward(
  address validator
) external nonReentrant onlyStakeManager
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`validator`	address	Validator address.

slash

Slash validator and transfer the slashed amount to `address(0)`.

Declaration:

```
function slash(
  address validator,
  uint8 factor
) external nonReentrant onlyStakeManager returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`validator`	address	Validator address.
`factor`	uint8	Slash factor.

FreeFee

Contents

Globals

Var	Type
userUsage	mapping(address => struct IFreeFee.UsageInfo)
totalUsage	uint256
globalTimestamp	uint256

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

estimateTotalLeft

Estimate total daily free fee left.

Declaration:

```
function estimateTotalLeft(
  uint256 timestamp
) public returns (uint256 totalLeft)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`timestamp`	uint256	Current timestamp

estimateUsage

Estimate user daily free fee usage.

Declaration:

```
function estimateUsage(
  struct IFreeFee.UsageInfo ui,
  uint256 timestamp
) public returns (uint256)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`ui`	struct IFreeFee.UsageInfo	User usage information
`timestamp`	uint256	Current timestamp

estimateFreeFee

Estimate user daily free fee left.

Declaration:

```
function estimateFreeFee(
  address user,
  uint256 timestamp
) external returns (uint256)
```

Modifiers: No modifiers

Args:

Arg	Type	Description
`user`	address	User address
`timestamp`	uint256	Current timestamp

consume

Consume user usage, it only can be called by router.

Declaration:

```
function consume(
  address user,
  uint256 usage
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

Args:

Arg	Type	Description
`user`	address	Transaction sender
`usage`	uint256	Usage amount

onAfterBlock

After block callback, it only can be called by router, it will update `globalTimestamp` if the time interval exceeds `freeFeeRecoverInterval`.

Declaration:

```
function onAfterBlock(
) external nonReentrant onlyRouter
```

Modifiers:

Modifier
nonReentrant
onlyRouter

UnstakePool

Contents

Globals

Var	Type
balanceOf	mapping(address => uint256)
totalSupplyOf	mapping(address => uint256)

Functions

constructor

No description

Declaration:

```
function constructor(
) public Only
```

Modifiers:

Modifier
Only

deposit

Deposit REI to `UnstakePool`, only can be called by stake manager, this will be called when user starts unstake.

Declaration:

```
function deposit(
  address validator
) external nonReentrant onlyStakeManager returns (uint256 shares)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`validator`	address	Validator address.

withdraw

Withdraw REI and burn shares, only can be called by stake manager, this will be called when unstake timeout.

Declaration:

```
function withdraw(
  address validator,
  uint256 shares,
  address payable to
) external nonReentrant onlyStakeManager returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`validator`	address	Validator address.
`shares`	uint256	Number of shares.
`to`	address payable	REI receiver address(this value is set when the user starts unstake).

slash

Slash validator and transfer the slashed amount to `address(0)`.

Declaration:

```
function slash(
  address validator,
  uint8 factor
) external nonReentrant onlyStakeManager returns (uint256 amount)
```

Modifiers:

Modifier
nonReentrant
onlyStakeManager

Args:

Arg	Type	Description
`validator`	address	Validator address.
`factor`	uint8	Slash factor.

FreeFeeToken

Contents

Functions

constructor

No description

Declaration:

```
function constructor(
) public AbstractToken
```

Modifiers:

Modifier
AbstractToken

balanceOf

No description

Declaration:

```
function balanceOf(
) public returns (uint256)
```

Modifiers: No modifiers

Config

Config contract for mainnet. In REI Network, system contracts can be updated through hard forks, and DAO logic will be added in the futhure.

Contents

Functions

stakeManager

No description

Declaration:

```
function stakeManager(
) external returns (address)
```

Modifiers: No modifiers

systemCaller

No description

Declaration:

```
function systemCaller(
) external returns (address)
```

Modifiers: No modifiers

unstakePool

No description

Declaration:

```
function unstakePool(
) external returns (address)
```

Modifiers: No modifiers

validatorRewardPool

No description

Declaration:

```
function validatorRewardPool(
) external returns (address)
```

Modifiers: No modifiers

fee

No description

Declaration:

```
function fee(
) external returns (address)
```

Modifiers: No modifiers

freeFee

No description

Declaration:

```
function freeFee(
) external returns (address)
```

Modifiers: No modifiers

feePool

No description

Declaration:

```
function feePool(
) external returns (address)
```

Modifiers: No modifiers

router

No description

Declaration:

```
function router(
) external returns (address)
```

Modifiers: No modifiers

contractFee

No description

Declaration:

```
function contractFee(
) external returns (address)
```

Modifiers: No modifiers

unstakeDelay

No description

Declaration:

```
function unstakeDelay(
) external returns (uint256)
```

Modifiers: No modifiers

withdrawDelay

No description

Declaration:

```
function withdrawDelay(
) external returns (uint256)
```

Modifiers: No modifiers

dailyFee

No description

Declaration:

```
function dailyFee(
) external returns (uint256)
```

Modifiers: No modifiers

dailyFreeFee

No description

Declaration:

```
function dailyFreeFee(
) external returns (uint256)
```

Modifiers: No modifiers

userFreeFeeLimit

No description

Declaration:

```
function userFreeFeeLimit(
) external returns (uint256)
```

Modifiers: No modifiers

feeRecoverInterval

No description

Declaration:

```
function feeRecoverInterval(
) external returns (uint256)
```

Modifiers: No modifiers

freeFeeRecoverInterval

No description

Declaration:

```
function freeFeeRecoverInterval(
) external returns (uint256)
```

Modifiers: No modifiers

feePoolLiquidateInterval

No description

Declaration:

```
function feePoolLiquidateInterval(
) external returns (uint256)
```

Modifiers: No modifiers

minIndexVotingPower

No description

Declaration:

```
function minIndexVotingPower(
) external returns (uint256)
```

Modifiers: No modifiers

setCommissionRateInterval

No description

Declaration:

```
function setCommissionRateInterval(
) external returns (uint256)
```

Modifiers: No modifiers

minerRewardFactor

No description

Declaration:

```
function minerRewardFactor(
) external returns (uint8)
```

Modifiers: No modifiers

getFactorByReason

No description

Declaration:

```
function getFactorByReason(
) external returns (uint8)
```

Modifiers: No modifiers

Using The Graph

Introduction

Indexing an application provides more efficient access to its organizational information. For example, search engines index the entire Internet, and you can search for information more easily through it

Indexing blockchain data is really, really hard.Blockchain properties like finality, chain reorganizations, or uncled blocks complicate this process further, and make it not just time consuming but conceptually hard to retrieve correct query results from blockchain data.

In addition, developers can build APIs (indexed "subgraphs") can used to query data specific to a set of smart contracts. Data is fetched with a standard GraphQL API. You can visit The Graph's documentation site to read more about The Graph protocol.

Due to the support of Ethereum tracing modules in REI network, The Graph is capable of indexing blockchain data in REI. This guide takes you through the creation of a simple subgraph for a Bank contract on testnet. This guide can be adapted for mainnet.

Using The Graph on REI Network

Checking Prerequisites

To use Graph on REI, you should run a local Graph Node against REI testnet and point your Subgraph to it. From the link below you can get information on how to run a nodeThe Bank Contract

Running a Graph Node on REI Network

For this example, a simple Bank contract will be used. You can find the Solidity file in the

REIBank-subgraph repository

The contract contains a bank where users can deposit rei for themselves, or transfer their rei in bank to others. They can also withdraw their rei from bank.

The main functions of the contract are the following:

- **deposit**() — function to send rei to the contract, then you can get your own proof of deposit in the bank
- **transfer**(address to, uint256 amount) — transfer your rei in bank to others
- **whithdraw**(unit256 amount) — redeem your money based on certificates of deposit

Events of the Bank Contract

The Graph uses the events emitted by the contract to index data. The bank contract emits three events:

- **Deposit**(address indexed from, uint256 indexed amount) — in the **deposit** function. It provides information related to deposit entry, including account address and deposit rei amount
- **Transfer**(address indexed from, address indexed to, uint256 indexed amount) — in the **transfer** function. It provides information on transaction between accounts in the bank, including accounts address and rei amount
- **Withdraw**(address indexed from, uint256 indexed amount) — in the **withdraw** function. It provides information related to withdraw entry, including account address and withdraw rei amount

Creating a Subgraph

This section goes through the process of creating a Subgraph. For the Bank Subgraph, a GitHub repository was prepared with everything you need to help you get started. The repository also includes the Bank contract, as well as a Hardhat configuration file and deployment script. If you are not familiar with it, you can check our Hardhat integration guide to learn about the configuration file and how to deploy a contract using Hardhat.

To get started, first clone the repository and install the dependencies

```
git clone https://github.com/bijianing97/REIBank-subgraph && cd REIBank-subgraph
npm i
```

Now, you can create the TypeScript types for the Graph by running

```
npm run codegen
```

The types will in the src/types/

> Creating the types requires you to have the ABI files specified in the `subgraph.yaml` file. This sample repository has the file already, but this is usually obtained after compiling the contract.

You also can use following command to specify the generated directory

```
npx graph codegen --output-dir xxxx
```

For this example, the contract was deployed to `0x1Ec9238A1c0adca222251e1f607a77237E8686a3`. The `README.md` file in the project has the steps necessary to compile and deploy the contract if required.

Subgraphs Core Structure

In general terms, Subgraphs define the data that The Graph will index from the blockchain and the way it is stored. The subgraph definition consists of a few files:

- **subgraph.yaml** — is a YAML file that contains the Subgraph's manifest
- **schema.graphql** — a GraphQL schema that defines what data is stored for your subgraph, and how to query it via GraphQL
- **AssemblyScript mappings** — AssemblyScript code that translates from the event data to the entities defined in your schema (e.g. `bank.ts` in this tutorial)

Schema.graphql

The schema for your subgraph is in the file `schema.graphql`. GraphQL schemas are defined using the GraphQL interface definition language. If you've never written a GraphQL schema, it is recommended that you check out this primer on the GraphQL type system. Reference documentation for GraphQL schemas can be found in the GraphQL API section.For this example, here one entry is defined for bank:

- **Account —** Record the users in bank, about their account address and balance

So the `schema.graphql` should look like the following snippet:

```
type Account @entity {
  id: ID!
  balance: BigInt!
  operateTime: BigInt!
}
```

Subgraph Manifest

The subgraph manifest `subgraph.yaml` defines the smart contracts your subgraph indexes, which events from these contracts to pay attention to, and how to map event data to entities that Graph Node stores and allows to query. The full specification for subgraph manifests can be found here.

For the example subgraph, `subgraph.yaml` is:

Some of the most important parameters in the `subgraph.yaml` file are:

- **repository** — Github repository of the subgraph
- **schema/file** — location of the `schema.graphql` file
- **dataSources/name** — the name of the Subgraph
- **network** — refers to the network name. This value **must** be set to the local graph node name which you setted
- **dataSources/source/address** — the address of the contract
- **dataSources/source/abi** — refers to where the interface of the contract is stored inside the `types` folder created with the `codegen` command
- **dataSources/source/startBlock** — refers to the start block from which the indexing will start
- **dataSources/mapping/file** — refers to the location of the mapping file, eg `bank.ts`
- **dataSources/mapping/entities** — definitions of the entities in the `schema.graphql` file
- **dataSources/abis/name** — where the interface of the contract is stored inside the `types/dataSources/name`
- **dataSources/abis/file** — refers to the location where the `.json` file with the contract's ABI is stored
- **dataSources/eventHandlers** — no value needs to be defined here, but this section refers to all the events that The Graph will index
- **dataSources/eventHandlers/event** — refers to the structure of an event to be tracked inside the contract. You need to provide the event name and its type of variables
- **dataSources/eventHandlers/handler** — refers to the name of the function inside the `mapping.ts` file which handles the event data

In short, the `subgraph.yaml` should look like the following snippet:

```
specVersion: 0.0.2
schema:
  file: ./schema.graphql
dataSources:
  - kind: ethereum
    name: Bank
    network: reidev
    source:
      address: "0x1Ec9238A1c0adca222251e1f607a77237E8686a3"
      abi: Bank
    mapping:
      kind: ethereum/events
      apiVersion: 0.0.6
      language: wasm/assemblyscript
      entities:
        - Account
      abis:
        - name: Bank
          file: ./abis/Bank.json
      eventHandlers:
        - event: Deposit(indexed address,indexed uint256)
          handler: handleDeposit
        - event: Transfer(indexed address,indexed address,indexed uint256)
          handler: handleTransfer
        - event: Withdraw(indexed address,indexed uint256)
          handler: handleWithdraw
      file: ./src/Bank.ts
```

Mappings

The mappings transform the Ethereum data your mappings are sourcing into entities defined in your schema. Mappings are written in a subset of TypeScript called AssemblyScript which can be compiled to WASM (WebAssembly). AssemblyScript is stricter than normal TypeScript, yet provides a familiar syntax.

The mapping file used for the Bank example can be found in the project:

```
export function handleDeposit(event: Deposit): void {
    const address = event.params.from
    const amount = event.params.amount
    const id = `${address.toHex()}`
    let instance = Account.load(id)
    if (!instance) {
        instance = new Account(id)
        instance.balance = amount
        instance.operateTime = BigInt.fromU32(1)
    } else {
        instance.balance = instance.balance.plus(amount)
        instance.operateTime = instance.operateTime.plus(BigInt.fromU32(1))
    }
    instance.save()
}
```

Deploying a Subgraph

With your local Graph Node, you can create your Subgraph executing the following code:

```
npx graph create <username>/<subgraph-name> --node <graph-node>
```

Where:

- **username** — refers to the username related to the Subgraph being created
- **subgraph-name** — the Subgraph name
- **graph-node** — refers to the URL of the hosted service to use. Typically, for a local Graph Node is `http://127.0.0.1:8020`

Once created, you can deploy your Subgraph by running the following command with the same parameters as before:

```
npx graph deploy <username>/<subgraph-name> \
--ipfs <ipfs-url> \
--node <graph-node> \
```

Where:

- **username** — refers to the username used when creating the Subgraph
- **subraph-name** — refers to the Subgraph name defined when creating the Subgraph
- **ipfs-url** — refers to the URL for IPFS. For your local Graph Node, the default value is `http://localhost:5001`
- **graph-node** — refers to the URL of the hosted service to use. For your local Graph Node, the default value is `http://localhost:8020`

The logs for the sucessful deployed should look like:

```
✔ Apply migrations
✔ Load subgraph from subgraph.yaml
  Compile data source: Bank => build/Bank/Bank.wasm
✔ Compile subgraph
  Copy schema file build/schema.graphql
  Write subgraph file build/Bank/abis/Bank.json
  Write subgraph manifest build/subgraph.yaml
✔ Write compiled subgraph to build/
  Add file to IPFS build/schema.graphql
                .. QmQUcxnQNfSTYohwtjqhDmZaLabiaBppB9yJSe3Swgm6TQ
  Add file to IPFS build/Bank/abis/Bank.json
                .. QmWQJTY12VjJC1AZrfF9FjDCCR6jdNakUXW95BXbMp4UWK
  Add file to IPFS build/Bank/Bank.wasm
                .. QmVrjrWSuFcnFRQ7kuoLkwyJg3VCbp6A8U62Z8XGrd6WFA
✔ Upload subgraph to IPFS

Build completed: QmcLXNBf4n29hPEzcLwaNtbRNPLojUKK9wwBUszPXdywLu

Deployed to http://localhost:8000/subgraphs/name/bank/graphql

Subgraph endpoints:
Queries (HTTP):     http://localhost:8000/subgraphs/name/bank
Subscriptions (WS): http://localhost:8001/subgraphs/name/bank
```

DApps can now use the Subgraph endpoints to fetch the data indexed by The Graph protocol.

If there has existed call record for contract, you can call the Graph node to get the data, just like this:

```
curl -X POST -d '{ "query": "{accounts{id,balance,operateTime}}"}' http://localhost:800
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   366  100   318  100    48  26500   4000 --:--:-- --:--:-- --:--:-- 33272
{
   "data" : {
      "accounts" : [
         {
            "balance" : "1999999999999999922243",
            "id" : "0x809fae291f79c9953577ee9007342cff84014b1c",
            "operateTime" : "3"
         },
         {
            "balance" : "150000000000000000000",
            "id" : "0x898b84b6a6430eed36a6cfc14a1cb7da326c91c4",
            "operateTime" : "1"
         },
         {
            "balance" : "77777",
            "id" : "0x8dd89ed567ac41866babddeb8931af2a695106af",
            "operateTime" : "0"
         }
      ]
   }
}
```

References

The Graph Docs
The Graph Docs

Running a Graph Node on REI Network

This document describes how set up a local graph node for REI network

Introduction

The Graph is a decentralized protocol for indexing and querying data from blockchains, starting with Ethereum. It makes it possible to query data that is difficult to query directly. Graph nodes which source events from blockchain to deterministically update a data store that can be queried via the GraphQL endpoint

There are two ways to launch a Graph Node: you can use Docker to run related images, or you can compile their Source code. The steps described in this document will only be run through docker, because it is more convenient, and you can launch a Graph Node very quickly.

> The steps described in this guide have been tested in both Ubuntu 20.04-based and MacOs environments, and they will need to be adapted accordingly for other systems.

Prerequisites

Before you start, you need some pre-environment configuration on your system:

- Docker
- Docker Compose

In this guide, you will learn how to run a Graph node against a node with the REI testnet node.This guide can also be adapted for mainnet.

Running a Graph Node

The first step is to clone the Graph Node repository:

```
git clone https://github.com/graphprotocol/graph-node/
cd graph-node/docker
```

Then execute the `setup.sh` file, this file will pull all the necessary Docker images and write the necessary information into the `docker-compose.yml` file.

```
./setup.sh
```

The log information returned looks like:

```
f635aec27645: Pull complete
a165c6729250: Pull complete
b0aa4f86b631: Pull complete
9efc4664d9d2: Pull complete
Digest: sha256:3162a6ead070474b27289f09eac4c865e75f93847a2d7098f718ee5a721637c4
Status: Downloaded newer image for postgres:latest
Pulling graph-node (graphprotocol/graph-node:)...
latest: Pulling from graphprotocol/graph-node
a10c77af2613: Pull complete
d0208cebd274: Pull complete
7763c68bd3a4: Pull complete
3a5e9447d0ef: Pull complete
cb1f41c6311: Pull complete
de9a00723392: Pull complete
Digest: sha256:da6a87c67b86941566052a633d849c3a409d549874ad40e0b93d55dbd316851a
Status: Downloaded newer image for graphprotocol/graph-node:latest
Creating docker_ipfs_1     ... done
Creating docker_postgres_1 ... done
Creating docker_graph-node_1 ... done
Starting graph-node ... done
Host IP: 172.18.0.1
Stopping docker_graph-node_1 ... done
```

When everything is ready, you should to revise the "Ethereum environment" inside the `docker-compose.yml` file, so that it points to the endpoint of the node you are running this Graph Node against. Note that the `setup.sh` file detects the `Host IP` and writes its value, so you'll need to modify it accordingly.

The entire `docker-compose.yml` file should look something similar to:

```
version: '3'
services:
  graph-node:
    image: graphprotocol/graph-node
    ports:
      - '8000:8000'
      - '8001:8001'
      - '8020:8020'
      - '8030:8030'
      - '8040:8040'
    depends_on:
      - ipfs
      - postgres
    environment:
      postgres_host: postgres
      postgres_user: graph-node
      postgres_pass: let-me-in
      postgres_db: graph-node
      ipfs: 'ipfs:5001'
      ethereum: 'reitest:http://127.0.0.1:3030'
      RUST_LOG: info
  ipfs:
    image: ipfs/go-ipfs:v0.4.23
    ports:
      - '5001:5001'
    volumes:
      - ./data/ipfs:/data/ipfs
  postgres:
    image: postgres
    ports:
      - '5432:5432'
    command: ["postgres", "-cshared_preload_libraries=pg_stat_statements"]
    environment:
      POSTGRES_USER: graph-node
      POSTGRES_PASSWORD: let-me-in
      POSTGRES_DB: graph-node
    volumes:
      - ./data/postgres:/var/lib/postgresql/data
```

Finally, you can start with the following command :

```
docker-compose up
```

```
root@rei-dev:/mnt2/graph-node/docker# docker-compose up
Creating network "docker_default" with the default driver
Creating docker_ipfs_1     ... done
Creating docker_postgres_1 ... done
Creating docker_graph-node_1 ... done
Attaching to docker_postgres_1, docker_ipfs_1, docker_graph-node_1
ipfs_1        | Changing user to ipfs
postgres_1    | The files belonging to this database system will be owned by user "postgre
s".
postgres_1    | This user must also own the server process.
postgres_1    |
postgres_1    | The database cluster will be initialized with locale "en_US.utf8".
postgres_1    | The default database encoding has accordingly been set to "UTF8".
postgres_1    | The default text search configuration will be set to "english".
postgres_1    |
postgres_1    | Data page checksums are disabled.
postgres_1    |
postgres_1    | fixing permissions on existing directory /var/lib/postgresql/data ... ok
postgres_1    | creating subdirectories ... ok
postgres_1    | selecting dynamic shared memory implementation ... posix
postgres_1    | selecting default max_connections ... 100
postgres_1    | selecting default shared_buffers ... 128MB
postgres_1    | selecting default time zone ... Etc/UTC
postgres_1    | creating configuration files ... ok
ipfs_1        | ipfs version 0.4.23
```

Wait for a while and you will see the log with the latest block synced :

```
0|graph-node  | Feb 08 13:13:24.257 INFO Syncing 1 blocks from Ethereum.
0|graph-node  | , code: BlockIngestionStatus, blocks_needed: 1, blocks_behind: 1, latest_block_head: 404804, curren
t_block_head: 404803, provider: reidev-rpc-0, component: BlockIngestor
0|graph-node  | Feb 08 13:13:27.272 INFO Syncing 1 blocks from Ethereum., code: BlockIngestionStatus, blocks_needed
: 1, blocks_behind: 1, latest_block_head: 404805, current_block_head: 404804, provider: reidev-rpc-0, component: Bl
ockIngestor
0|graph-node  | Feb 08 13:13:31.289 INFO Syncing 1 blocks from Ethereum.
0|graph-node  | , code: BlockIngestionStatus, blocks_needed: 1, blocks_behind: 1, latest_block_head: 404806, curren
t_block_head: 404805, provider: reidev-rpc-0, component: BlockIngestor
0|graph-node  | Feb 08 13:13:33.300 INFO Syncing 1 blocks from Ethereum., code: BlockIngestionStatus, blocks_needed
: 1, blocks_behind: 1, latest_block_head: 404807, current_block_head: 404806, provider: reidev-rpc-0, component: Bl
ockIngestor
0|graph-node  | Feb 08 13:13:36.315 INFO Syncing 1 blocks from Ethereum., code: BlockIngestionStatus, blocks_needed
: 1, blocks_behind: 1, latest_block_head: 404808, current_block_head: 404807, provider: reidev-rpc-0, component
0|graph-node  | , : BlockIngestor
0|graph-node  | Feb 08 13:13:38.334 INFO Syncing 1 blocks from Ethereum., code: BlockIngestionStatus, blocks_needed
: 1, blocks_behind: 1, latest_block_head: 404809, current_block_head: 404808, provider: reidev-rpc-0, component: Bl
ockIngestor
```

As you can see, you have a Graph Node running against the REI TestNet. This example is also suitable for mainnet

References

The Graph Docs
The Graph Docs

Using Gnosis Safe

Gnosis safe wallet is now live: https://gnosis.rei.network

Introduction

Gnosis Safe is the most trusted platform to manage digital assets for Ethereum compatible chains providing bellow features.

Multi-signature

Fully customize how you manage your company crypto assets, with the option to require a predefined number of signatures to confirm transactions. Require multiple team members to confirm every transaction in order to execute it, which helps prevent unauthorized access to company crypto.

Assets Management

Gnosis Safe supports Native Token(REI), ERC20 (Tokens) and ERC721 (NFTs). You can also see the fiat values of your assets.

Usages

You can setup a Safe at https://gnosis.rei.network in 60 seconds.

You specify the number of owners and their owner account addresses. Compatible account addresses include hardware wallets, Metamask, Authereum, WalletConnect enabled wallets, and others. You can see the asset overview and can make transactions such as custom transactions for direct smart contract interaction. ENS names are supported.

Exchange Integration

REI Network is **compatible** with all data structures and RPC interfaces of Ethereum, so there is almost no difference between access to REI Network and Ethereum blockchain. The only difference is that in the finality of the block, there is **no block reorganization** in REI Network, the block will never revert as it's generated, which can simplify the process of dealing with transaction status

API Reference

What is REI DAO?

REI DAO is a decentralized governance community, a manifestation of the REI Network blockchain spirit of being more decentralized, open and transparent in its governance model, and returning governance to each `$REI` holder.

The organization is composed of all `$REI` token holders, community members and ecological stakeholders.

Vision

REI Network, as the on-chain infrastructure, will give the community more management authority.

> **REI Network+DAO** is not limited to governance, but will also be reflected in the return of the Foundation's decision-making authority to the community. For example, the foundation will only be responsible for setting the rules for Grants, and it will be up to the `$REI` holders to decide whether new projects applying for REI Network Grants should be approved and how much to award.

REI Network will create a positive interaction with `$REI` holders. In the traditional Web2 platform, the value of network effect growth is captured by the platform.

As a gateway to team collaboration and productive relationship innovation (Web 3.0), the network effect of DAO is stronger than centralized Web2, and its value is distributed by the platform to community members based on the level of contribution, thus further enhancing the impact of the network ecosystem and the infrastructure needed for ecological building.

A more robust infrastructure will attract more members and provide stronger value support for tokens on the REI network.

What is Foundation and how does it relate to REI DAO?

- The mission of the Foundation is to plan the development direction of the REI network accordingly and ensure that REI DAO can survive as a fully decentralized organization. It ensures the stable operation of the chain and the stable development of the early ecology, thus slowly transitioning to a collaborative governance model and eventually realizing a decentralized governance model for the REI network.
- The Foundation has hired core personnel to develop a strategy to achieve the core roadmap, and according to the Foundation's plan, decentralized governance is a key goal for the future and is currently underway. While the Foundation is part of the REI DAO community, the Foundation does not hold a privileged position in the community and the REI network itself is governed through the voting rights of `$REI` holders.

Can foundations vote in REI DAO governance?

- The Foundation will gradually transfer control of the remaining 250 million `$REI` Eco-Development Fund to REI DAO, so the Foundation does not control or participate in REI DAO governance voting.

What is the legal structure of the Foundation?

- Non-profit organization registered in Singapore

Background

Why REI Network need DAO?

DAO is a kind of spontaneous governance method by the community. Since 2019, DAOs with different functions have gradually emerged. Today, DAO is constantly exploring its utility and value in the realization of diversified purposes, multi-scenario applications, and ecosystems. DAO is the basic entry and interpretation in the Web 3.0 era, and it is also a concrete scene of a decentralized and discrete governance model based on blockchain.

The era of Web 3.0 is not only a necessary step for REI Network to transform itself as a technology carrier, but also as a social experiment in the crypto world itself needs to take a step forward.

Why can REI Network implement DAO?

The REI Network originated from the community construction of GXChain. It has been an important part of the public chain ecosystem since the establishment of GXChain, and it is also one of the largest and most active blockchain communities in the world. It includes core protocol developers, crypto-economic researchers, cypherpunks, node operators, `$REI` holders, application developers, and community users.

With the continuous precipitation and development over the years, the total coverage of the population has reached hundreds of thousands of people. The REI Network community currently covers more than 20 countries around the world, and the international community includes areas that are extremely popular in the crypto economy, such as South Korea, India, Spain, Russia, Africa, Europe, and Southeast Asian countries.

In addition, the REI Network community also covers the aggregation community including the REI super node ecosystem, which is also one of the most active blockchain enthusiasts in the crypto world. The REI Network community has rich user portraits, super nodes, partners, ecosystem projects, and endogenous community KOLs.

At the same time, carrying the origin of REI Network community governance, GXChain will continue to build a larger and more differentiated crypto community in the future, deepening the consensus of REI Network and inspiring strong community governance and ecosystem construction ability.

> The above objective conditions give us reason to believe that the conditions for realizing DAO autonomy on an EVM-compatible public chain with a broad community base and vitality are ripe, and the era of REI DAO has come.

What can DAO bring to REI Network?

Let the community vote on how to deploy their funds to support builders and developers for their own development. Community orientation inspires collective intelligence and empowers DAO to make decentralized decision-making capabilities that may outperform a single decision-maker.

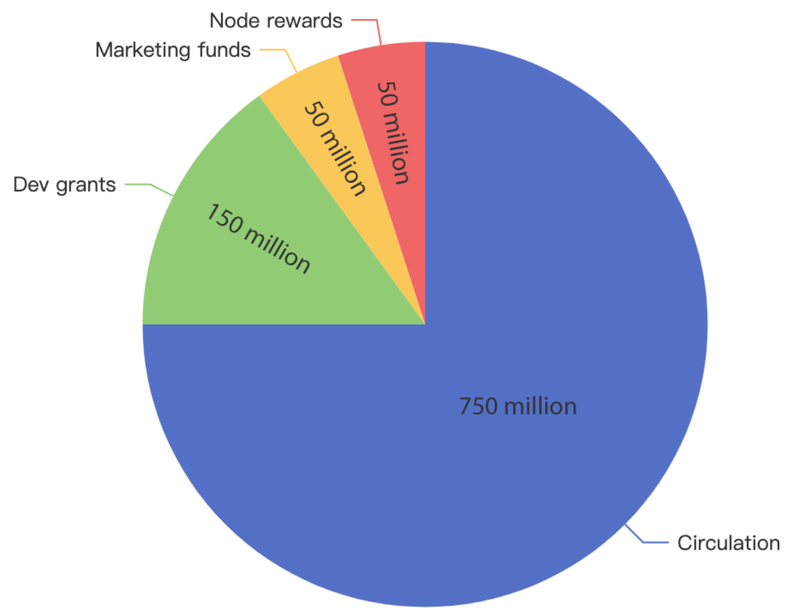
DAO is a form of management organization with specific implementation methods for promoting community development. Compared with the community autonomy and proposal mechanism in the GXChain era, DAO is a more standardized, decentralized, and diversified management form, which can not only objectively and effectively reflect the actual needs of the community, but also the implementation plan that is compatible with the opinions of various sectors.

Management content of REI DAO

REIcosystem & Marketing

As mentioned in "#Governance Proposal: From GXChain 2.0 to REI Network", the unlocked part of the team is upgraded to an "Ecosystem Grants". The team took out all the unlocked 25 million $GXC (250 million `$REI`) as Ecosystem Grants, and the specific distribution is as follows

- **50 million** `$REI` for marketing
- **150 million** `$REI` is used to incentivize developers and the ecosystem.
- **50 million** `$REI` for node rewards



In the above three parts, REIcosystem & Marketing will involve the marketing promotion of the team and the cost of ecosystem construction, including but not limited to the following parts:

- **Grants distribution:** Grants are designed to support projects built on the REI Network to promote the improvement of the REI Network ecosystem. REI Network will specifically evaluate Grants based on the project's influence, development prospects, team experience, and ecosystem relevance to decide the quantity of distribution.
- **Market promotion:** For the ever-changing REI Network, it is necessary to maintain a high degree of industry acumen at all times. In the market and cooperation, transaction activity, team building, and content creation, the market recognition and global popularity of REI Network will be comprehensively promoted. Value return and other aspects ensure an efficient value transfer model.
- **Developer incentives:** At the developer level, more incentives from community developers need to be given to promote their output, including the development of tool products, technical improvements, positive feedback, etc.

Apart from the specific payment method, which needs to be realized by REI DAO, all the expenses for marketing, operation, and ecosystem development are managed and spent in the way of DAO, and the distribution of node rewards is a linear process.

Advantage orientation of DAO governance

> ⓘ Governance based on REI DAO will bring the following changes:

Comprehensive and transparent community management.

The management method of REI DAO is all-around and transparent. Under the management of DAO, the expenditure of Marketing grants and REIcosystem / dev-grants will be carried out on a daily basis under the supervision and management of the community.

Distributed opinion collection mode.

The proposal submission process is different from the previous node governance method and the management method of the proposal proposed by the council. Under DAO governance, as long as a certain amount of `$REI` holding qualifications are met, a governance proposal can be put forward, so governance demands expression under these conditions will become flattered and distributed. On this basis, community opinions about future directions including protocol deployment, grants distribution, market budget allocation, and evolution route from REI DAO can be more efficiently and directly proposed in the form of "proposals".

Realization of REI Network governance value.

Decentralized governance through DAO encourages open participation in the ecosystem development of the REI Network. By holding governance tokens, participants not only have the opportunity to share the rewards associated with the marketization of REI Network, but also have the incentive to define where the system will go, and the economic and social impact it will have.

How the REI DAO is implemented ?

Multisig wallet: security and openness

The team uses the Genosis multisig wallet to ensure the centralization and transparency of the DAO. The above-mentioned funds will be implemented through the technical implementation of multi-signature wallets to realize the specific expenditure of funds under the jurisdiction of DAO. By using the Genosis multisig wallet, there will be multiple people jointly managing the funds and will satisfy both security and decentralization features.

75% principle:

The use of funds requires a multi-sig wallet, which is suitable for the decision-making scenario of the muti-people or institution. REI Network decided to use the multisig wallet management to implement distributed management of the overall capital flow. It minimizes the risk of asset loss when a single private key is lost, and the use of this part of the funds must be approved by 3/4 of the following four addresses to take effect.

The specific addresses with administrative rights are as follows:

- 0x302...9e6D
- 0x8cD...E092
- 0xA12...FFe2
- 0x524...4782

How REI DAO works ?

Proposal Generation

1.Community proposals generate votes:

Required qualifications: REI Network governance 50,000 `$REI`, or collects an equal number of proposal support.

2.Proposals produced by the Council members:

2.1 REI DAO Council

The Council is a community-based organization specially established to promote the prosperity of on-chain governance. In this organization, in order to more efficiently promote the actual governance utility of DAO, the first batch of REI DAO Council members will be invited by the team and will be generated within 10 working days.

2.2 Proposal power of Council members

- Council members can directly put forward proposals.
- The first step in participating in governance is to initiate any discussion that benefits the REI network and build influence in the community to get more people to agree with your proposal.
- Then get in touch with Council members to discuss your proposal on the Community Call and Twitter Spaces. After making a formal proposal, it will be upgraded to the voting session.

> **Proposal language**
>
> We try to keep the discussion of the content of the proposal in English. While we welcome discussions on REI Network development in other languages, such as Korean, French, Chinese, Spanish, etc. (eg, in REI Forum), English is currently preferred due to the consistency of discussion.

Types of proposals

REI Network operating mechanism proposal

E.g:

- Modify the dynamic parameters of the blockchain, such as block size, block interval, and other smart contract creation and call rates, and modify the dynamic parameters of the blockchain, such as block size, block interval, etc.;
- The block reward of trust nodes, the number of trust nodes and the number of active council members, etc.;
- Transfer, issuance of assets, and various transaction fees;
- Smart contract creation and call rates, etc.
- Adjustment at the strategic level of REI Network: change the project name, adjust the technical structure, adjust the roadmap, etc.

Fund application proposals

- Including ecosystem construction, team building, developer incentives, REI Network marketing, public welfare donations, etc.
- Applying for proposals with different amounts will involve different voting approval rates and voting durations.

Proposal voting process

Voting duration:

Once the proposed proposal meets the above requirements, the REI DAO begins a 7-day voting period. Ongoing discussions are conducted in the REI DAO Forum. If the proposal is successfully passed, there will be a two-day lock period before the proposal code is executed, and then the relevant proposal content will be finally executed.

Turnout requirements:

In order to get the vote passed, it is generally required that at least a certain amount of `$REI` (2%-10%) must vote in favor. The purpose of this quorum is to ensure sufficient legitimacy and actual participation;

If the following situations are involved, the minimum approval rate needs to be increased to 10%

- The content of the proposal involves the evolution of the major strategic level of the project: for example: changing the project name, adjusting the technical structure, splitting tokens, significantly adjusting the evolution route of the roadmap, etc.
- The total amount of REI tokens requested to be used must be more than 5% of the total number of tokens.

Initial principle:

The content mentioned in this part is the initial setting of the on-chain governance section, and the specific setting of this part is also within the scope of the operating mechanism proposal.

> Good to know : The following content statement is an initial value setting and the turnout requirement can be modified in the future through community governance.

REI Network operating mechanism proposal	Minimum approval rate required for proposal	Voting duration
On-chain parameter modification	5%	7 DAYS
Transfer, issuance of assets, and trading fees	5%	7 DAYS
Staking and quota adjustment	5%	7 DAYS
REI Network strategic adjustment	10%	10 DAYS

Grants/Marketing application type proposal	Minimum approval rate required for proposal	Voting duration
Under $100,000	2%	3 DAYS
$100,000-200,000	3%	5 DAYS
$200,000-400,000	3%	7 DAYS
Over $500,000	5%	7 DAYS
More than 5% of the total number of tokens	10%	10 DAYS

Principles of Proposal Implementation

Orderly execution of proposals:

When multiple on-chain proposals or tasks need to be executed, the executor should take into account the enthusiasm of REI Network development and the limited timeliness of the proposal execution content to execute or approve the corresponding funds

Not against the fundamental interests of REI Network:

All on-chain proposals should be for the purpose of actively promoting the development of REI Network. Under this premise, all proposals will be considered invalid if there is the malicious use of corresponding funds or any part that is not related to the development of REI Network.

Force Majeure:

In the process of proposal, voting, and implementation, if force majeure factors arise in the location of the participants, such as war, natural disaster, political control, etc., the proposal link should be postponed or carried out on another day as the case may be.

Do not violate local laws and public order and good customs:

In the use of REI Network funds, it should abide by local laws and the public order and social ethics of the place of use, so as to avoid unnecessary conflicts and brand losses.

Security mechanisms implemented by REI DAO:

a. The governance security mechanism mitigates or eliminates the adverse impact of harmful proposals on the REI network or the REI DAO community by setting a time delay before the proposal is formally implemented.

b. Before new code is deployed to REI Network, there is a 24-hour publicity period, during which all community members can review the code to confirm the security of REI Network. If the code is reviewed and deemed to have any adverse effect on REI Network, the deployment will be canceled immediately.

About Governance

Introduction

Governance is the core proposition of a chain, and a chain with decentralized governance will have the longest life.

- REI Network introduces both people and code into the complex governance system of the public chain through the governance method of voting and democratic consultation and discussion, thus ensuring the effectiveness of governance while achieving decentralized governance.
- Every `$REI` holder has the right to participate in the decentralized governance, and the operation and development direction of the public chain ecosystem will be decided by all `$REI` holders through democratic consultation and voting.

> **Governance Principles**
>
> REI Network will follow the principles of scientific governance, and will be governed through democratic consultation and discussion and voting, gradually transitioning to fully decentralized management.

Governance Process

Step1: Community consensus

The first step in participating in governance is to initiate any discussion that benefits the REI Network and to build influence in the community so that more people agree with your proposal.

Join the REI Network (GXChain) Discord Server!
Discord

Discord

Step2: Consensus check

You will need to create a forum proposal thread in the Consensus Check section while adhering to the forum rules, use your influence to let more community members know about your idea, create a poll in the proposal feedback, and receive timely community feedback to get the final proposal.

Consensus Check
REI Network

Forum

Step3: Proposal review

Proposals will be reviewed by the REI DAO Council.

What is the REI DAO Council?

The Council is composed of 21 deliberators, and the first members will be invited from nodes, community members, pioneers, and project parties who have contributed to REI Network before, and later will be decided by the Council itself. The discussion process will be moderated and recorded by the council members in three sessions: motion presentation, motion review, and motion vote.

Introducing the Governance Council
REI Network

Details

Step4: Voting

Provide voting tools and multiple voting strategies for surveying community sentiment on specific proposals or parameters, allowing every coin holder to participate more fairly in community governance.

Voting with Snapshot

Step5: Security mechanisms

- The governance security mechanism mitigates or eliminates the negative impact of harmful proposals on the REI Network or REI DAO community by setting a time delay before formal implementation of the proposal.
- New code is reserved for a 24-hour public notice period before deployment to REI Network, during which time all community members can review the code to confirm the security of REI Network. If the code is reviewed and is deemed to have any negative impact on REI Network, the deployment will be immediately cancelled.

Voting with Snapshot

Introduction to Snapshot

One of the most important aspects of a DAO is collective decision making. We'll be participating in exactly that by voting on a proposal with Snapshot!

What Do I Need?

Before we get going, let's make sure you've got everything you need. In order to vote with Snapshot, you will need:

- An wallet, such as **MetaMask**
- Governance tokens from the project you want to vote on,such as `$REI`

Voting & Snapshot

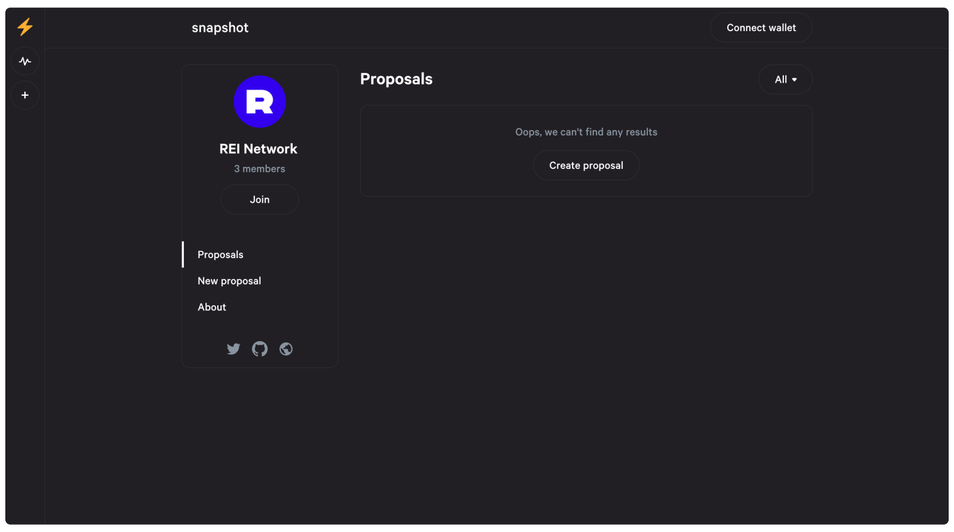
Snapshot is a decentralized, gasless voting system. It allows DAO members to vote on proposals with governance tokens like `$REI` , NFTs like Bored Ape Yacht Club, and more. Thanks to Snapshot's off-chain, gasless nature, it removes loads of friction from the voting process and makes it as easy as posting a Twitter poll.

Alright, let's cast a vote!

The Guide

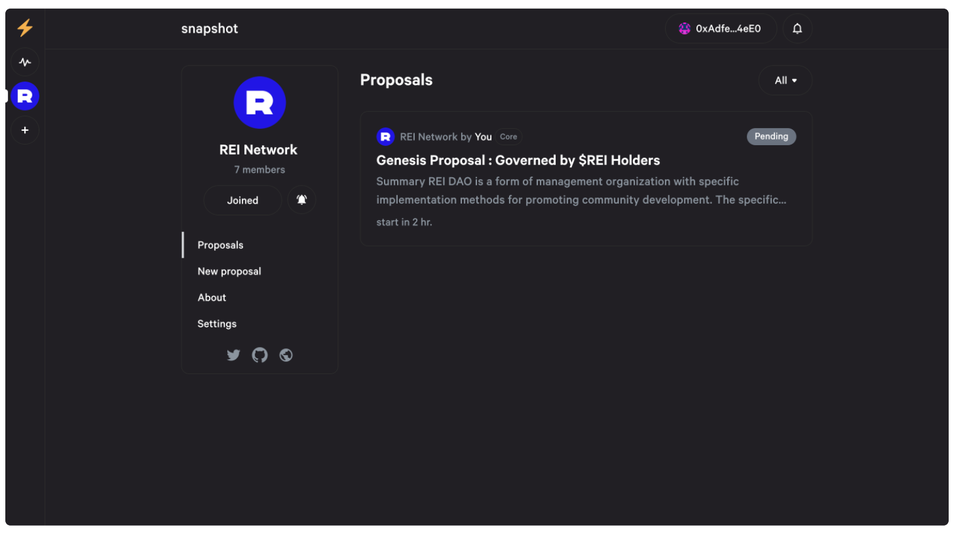
Step 1

Let's begin by heading to **https://snapshot.org/#/rei-network.eth** ; And `Connect wallet` with the button in the top-right corner.



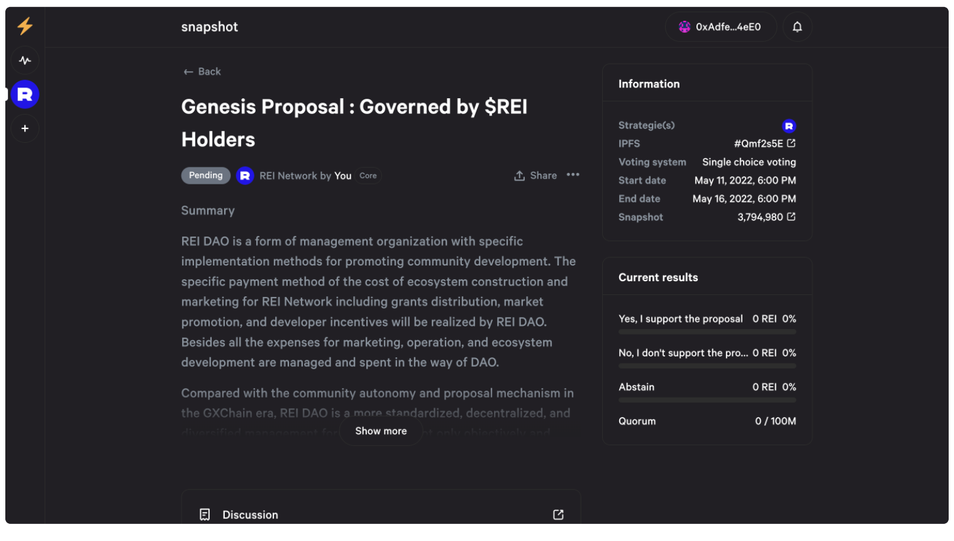
Step 2

Click `join` ; Congratulations, you have successfully joined the REI Network space!



Step 3

On the Proposals tab, find an active proposal that you wish to vote on. After familiarizing yourself with the proposal, cast your vote and click Vote. You will be prompted to confirm your vote, and if everything looks good, click Vote again.



Step 4

Finally, confirm your vote by signing it with your wallet. Although you are using your wallet, there are no gas fees associated with voting!

Congrats!

You have successfully participated in the governance of REI DAO !

FAQ

Frequently Asked Questions about REI DAO Governance

What is democratic consultation and discussion?

- Democratic consultation discussion refers to the process of gathering feedback on community suggestions prior to voting, and anyone can participate in democratic discussion of governance, as can anyone who does not hold any `$REI`.

What is Voting ?

- Voting is the process of voting on proposals for implementation, and anyone holding `$REI` can participate in voting on these proposals.

Does anyone have the power to initiate proposals?

- Yes, anyone has the power to initiate proposal discussions, instantly without holding any `$REI`.

What is the difference between Forum voting and Snapshot voting?

- Forum voting to measure the consensus and voice of the current proposal.
- Snapshot voting is the tool for performing proposal voting, with proposal voting governance by all `$REI` holders.

How do proposals initiated in the forum make it to the governance voting process?

- The council of deliberators will review the proposals democratically and fairly, following the governance principles of REI DAO.
- After the proposal has been reviewed and approved, the proposal will be submitted by a representative of the deliberators to initiate a vote for governance in Snapshot.

Can anyone create a governance poll?

- No, only members of the Council of Deliberators have the authority to create governance proposals at Snapshot.

What is the Council of Deliberators?

- REI DAO Council is composed of core team members, SuperNode and active community participants. The Council currently consists of 9 members and has the power to submit proposals for REI Network parameter changes, technical development directions and ecological bounty allocations.

What permissions are granted to my wallet address when Snapshot connects to Metamask?

- Wallet Address Balance: `$REI`

Do I need to pay a gas fee every time I vote?

- Voting in Snapshot, without paying any Gas.

Dashboard

> **Good to know:** The dashboard provides more information about the REI Network, such as charts for Total stake, Total gas stake , Total voting stake , Amount Of Gas Saved For Users . You can also find out the latest REI prices, REI token distribution, etc.

Navigate to:

- https://dao.rei.network/#/dashboard

Explanation of the terms in the Trends chart

- Total stake: Sum of total voting stake and total gas stake.
- Total voting stake: Sum of the number of votes in node validation
- Total gas stake: Sum of the number of REI staked in order to obtain free crude

Become a Validator

A validator is a participant in the network who runs validator nodes to run and secure the network. Here are some steps to run a validator node in REI Network:

Step1: Setup a validator node

Following quick start to setup your nodes and make sure your validator node is fully synced.

Quick Start

Minimum system requirements

- RAM: 16GB
- CPU: 4-core
- Storage: 500G SSD
- Bandwidth: 10Mb/s

Recommended system requirements

- RAM: 32GB
- CPU: 8-core
- Storage: 1T SSD
- Bandwidth: Unlimited

Step2: Submit a Validator Profile

This repository (repo) provides the details about offchain informations of validators.

GitHub - REI-Network/rei-validator: OffChain informations of REI network validator
GitHub

REI Validator Profile Guideline

Step3: Vote/Stake to your node

You can stake on REI Network by following the guides below:

Voting for a Validator

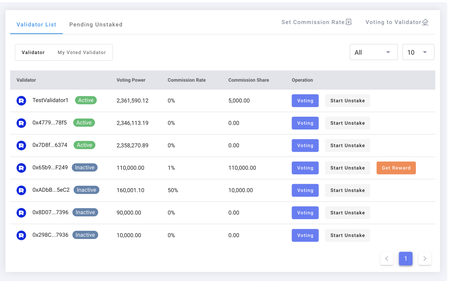
Step4: Set commission rate

Commission rate represents the proportion of the validator's reward, which is a number from 0 to 100. It can only be set by the validator, and the interval between 2 modifications cannot be less than **24 hours**.

> Eg: If a validator sets the Commission rate to 40, then 40% of the rewards he receives will be distributed to all users who voted for this validator.

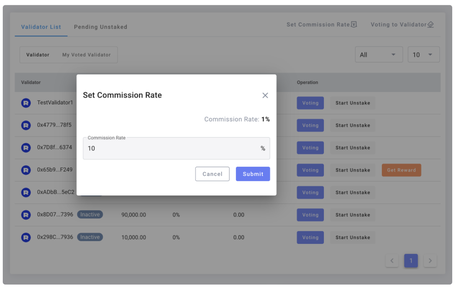
UI | Ethers

- Navigate to https://dao.rei.network/#/stake
- If the address in your wallet is a validator address, there will be a `Set Commission Rate` button appears beside `Voting to Validator`



Set Commission Rate

- Click on `Set Commission Rate` and enter a percent number you wanna share with voters

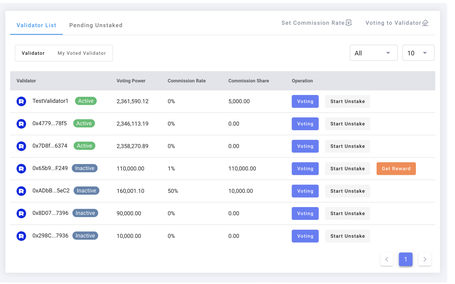


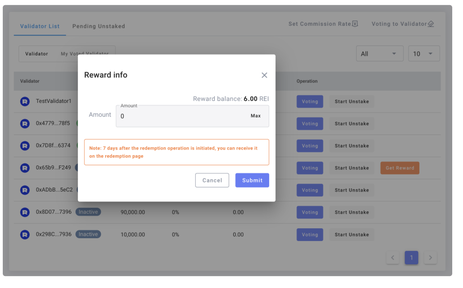
Set Commission Rate

Step5: Claim validator reward

UI | Ethers

Click on `Get Reward` and Claim block producer rewards in the validator list





> Node: The reward will be vested for 7 days before you can finally claim it, checkout Claim Vested Stake

Jail Mechanism

Introduction to REI Network Jail Mechanism

Since the launch of the REI Network super nodes, the network has been running stably for nearly 90 days. With the active participation of community users and various governance roles, the total number of staked votes has exceeded 150 million. Each validator has made great contributions to the stable operation and security verification of REI Network. However, in the process of long-term operation, some problems also appeared.

During node operation, due to the uneven hardware facilities and operational capabilities of validators, some validators may miss blocks or even stop producing blocks, which affects the overall performance and network security of the REI Network to a certain extent.

To ensure validators do not abuse their power, the Jail Mechanism will be introduced to solve the problems of block loss and other node performance instability of validators.

What is the Jail Mechanism?

If the "super node" misses blocks ≥ 300 in one day, it will be thrown into jail.

Description

Penalties in the Jail:

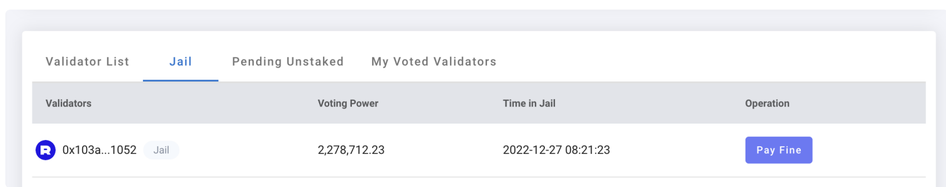
During this period, block production will be prohibited, and validators can choose to pay a fine of 20,000 REI to re-participate in block production. The fined 20,000 REI will be directly locked in the black hole address, and permanently cannot be withdrawn. Validators can also choose to withdraw all tokens and quit the node election, during this period, the tokens and node rewards staked by users and nodes will not be affected, and they can be freely withdrawn or reinvested in other nodes.

Features

- Real-time monitoring of validator block production status
- Re-participate in block production and share the rewards after paying the fine
- Validators can quickly query fine payment history

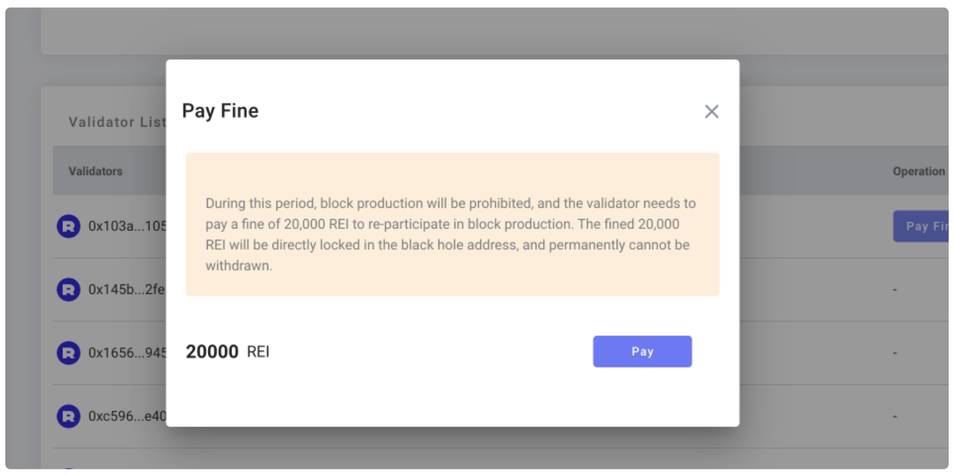
Which functions related to the Jail Mechanism does REI DAO have?

1. Validators and community users can check which validators have been Jailed in the Jail list



Jail List

2. The validators can quickly pay the 20,000 REI penalty and re-participate in block production



Pay Fine

3. Validators and community users can query the Jailed history of the current validator

History of Jail

Summary

The introduction of the Jail Mechanism will solve the problem that the continuous block loss of validators which affects the network stability and supervise the validator operators to further optimize the status of validators and improve the level of operation and maintenance, and further improve the security and stability of the chain.

Slash Mechanism

Introduction to REI Network Slash Mechanism

The consensus algorithm of the rei-network is `reimint`. In order to finalize the next block, `reimint` will collect votes from the top 21 validators for each candidate block. When the voting power collected by a certain candidate block is greater than 2/3 of the total voting power of the top 21, the block is produced. It is conceivable that if a validator signs multiple candidate blocks at the same height, the final result may finalize multiple blocks. In the worst case, it means that the blockchain has forked, and at this time all transactions may be rolled back to the height at which the fork started, and the status of all transactions cannot be determined.

Assuming that if the current block height is 100, and the confirmation height of the exchange deposit is 10, the attacker initiates a recharge to the exchange at the height of 100, and is confirmed by the exchange at the height of 110, and then the attacker immediately sells all the tokens into stablecoins on the exchange and withdraw them to other blockchains. If the attacker successfully launches a multi-signature attack at a height of 100 to roll back all subsequent transactions, the final result is that the tokens are still on the attacker's account, but the attacker managed to steal the equivalent of stablecoins from the exchange.

In order to prevent this from happening, multi-signature validators must be punished. All validators in the network (block-producing validators and non-block-producing validators) will continuously collect all votes, and once a validator is found to perform multi-signature, it will be packaged all repeated signatures to generate a proof, and then broadcast this proof to the whole network. When packing a block, the block-producing validator will pack the evidence it currently collects into the block header. When processing a block, the proof recorded multisig validators are slashed.

At present, the single slashing ratio set by the rei-network mainnet is **20%**. This means that all users who voted for the slashed validator and the validator itself will lose 20% of the REIs.

Proof

The first proof on rei-network appeared at the block height of 9185063. After decoding the extraData, the following two repeated votes can be obtained (for the convenience of observation, it has been converted to JSON format)

Vote 1:

```
{
  "chainId": 47805,
  "type": 1,
  "height": 9185061,
  "round": 0,
  "hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "index": 16,
  "signature": "0xa82def5f576359ebf9e84472df4e5e549b87127e4e0c8ddd40766da8354c44114ccfd
}
```

Vote 2:

```
{
  "chainId": 47805,
  "type": 1,
  "height": 9185061,
  "round": 0,
  "hash": "0x5a44cc5c7221c84499329e3cf383114d96d2f155d3de5a45bbe8917861d9fcdf",
  "index": 16,
  "signature": "0xe90523621aeab80466cadde79acc5b9ab1e35540b862071de77c83f69a7f507846b34
}
```

The meaning of each field:

Name	Meaning
`chainId`	Chain ID
`type`	Vote type, 1 means `prevote` type vote
`height`	Candidate block height
`round`	Number of rounds, if no consensus is reached in this round, it will enter the next round of voting, and the number of rounds will be increased by 1
`hash`	Candidate Block Hash
`index`	The position of the signing validator among the top 21 validators
`signature`	`secp256k1` signature, according to this signature and other fields, the address of the signing validator can be recovered

FAQ

Q: Why is the height of the block different from the height of the signature, one is 9185063 and the other is 9185061?

A: Because it takes time from the emergence of multi-signatures to the proof being packaged

Q: Why is the hash of one of the votes is `0x00....0000`?

A: Because in `reimint`, if the validator votes for `0x00....0000`, it means that the validator wants to enter the next round of voting. For many reasons, such as the validator doesn't receive this round of proposed blocks in time

Possible reasons for a Validator to be slashed

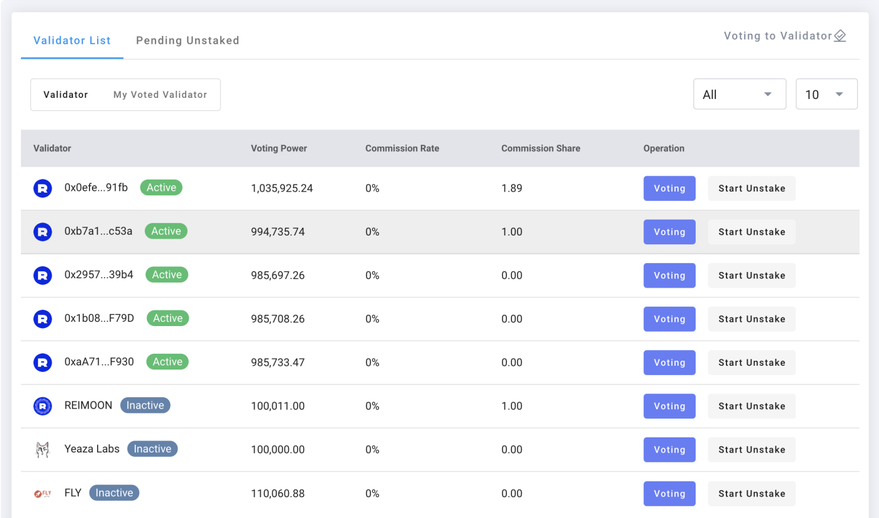
- Run two validators with the same address at the same time
- The validator's private key leaked
- Malicious tampering validator program

Voting for a Validator

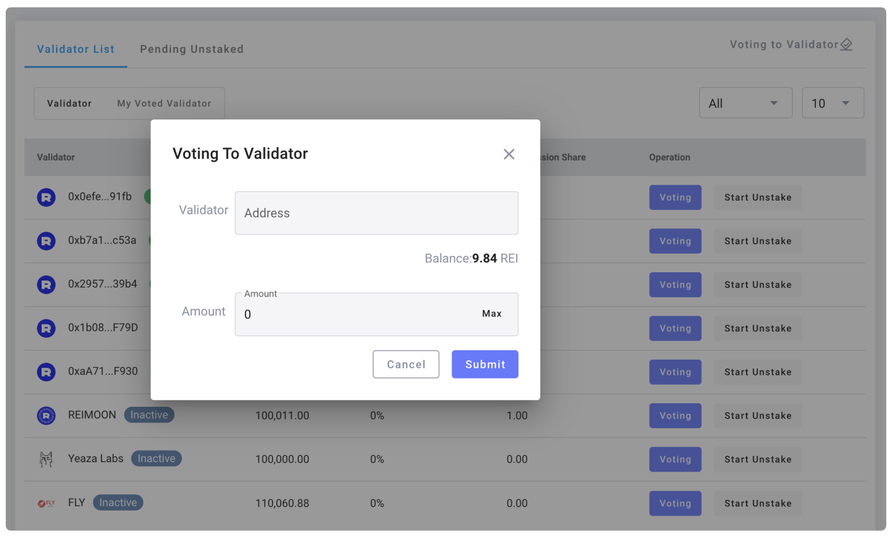
Vote/Stake to a Validator

UI | Ethers

Step1: Navigate to https://dao.rei.network/#/stake



Step2: Click on `Voting to Validator`, enter validator `address` and submit the `amount` you want to stake/vote

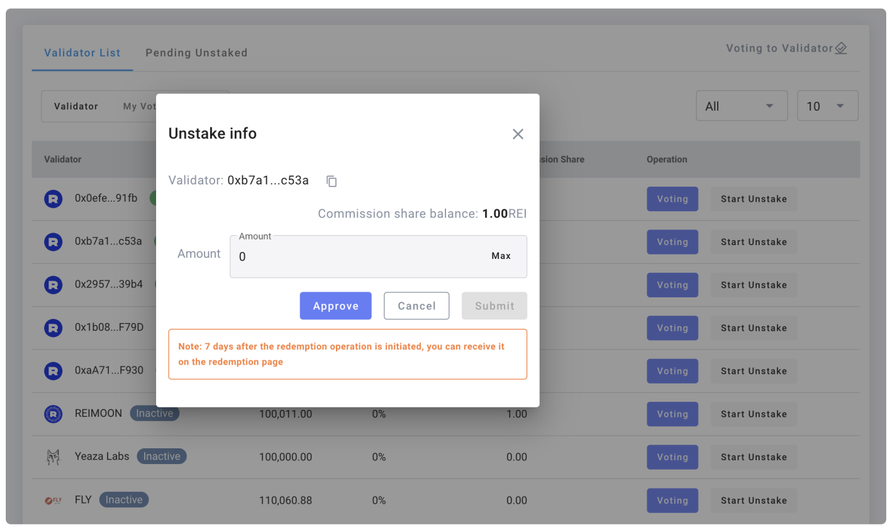


> **Why am I not able to Stake?**
>
> Check if you have `REI` in your wallet on REI Network
>
> **I've staked to 0x...abcd, but not record find in ACTIVE VALIDATOR?**
>
> Because only the 21 nodes with the most votes are currently displayed in the list of activated nodes

UnVote/UnStake

UI | Ethers

Once you log in you will find this on the Validator of the Staking Dashboard. Here you will see an `Unstake` button for each of the validators. Click on the `Unstake` button for whichever validator that you had staked to.



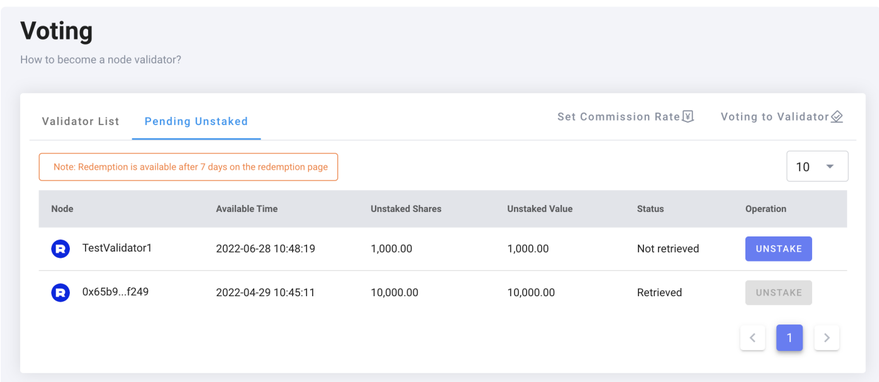
> Note: There will be a **vesting period** after unvote/unstake, find the pending unstake record in `Pending Unstake` Dashboard

Claim Vested Stake

UI | Ethers

You can find unstake list in `Pending Unstaked` Dashboard.

Once the vesting period is completed, the `UNSTAKE` button will be enabled and you can then claim your staked tokens to your wallet now.



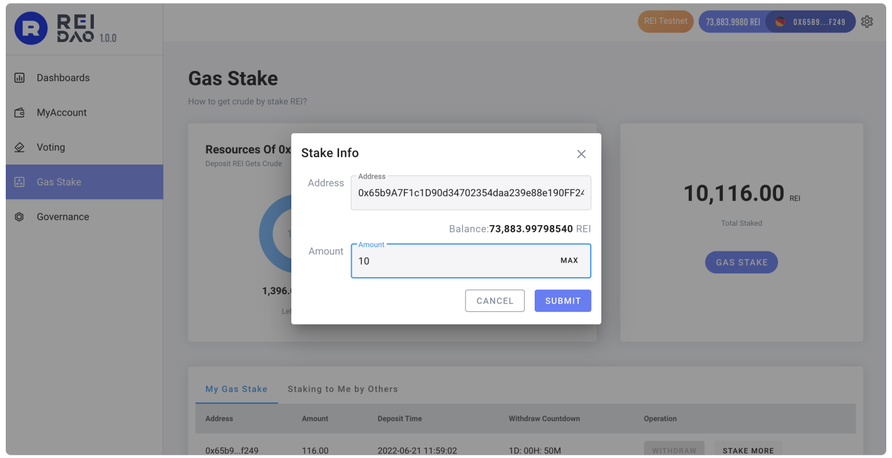
Stake for Free Gas

Stake for free gas

UI | Ethers

Step1: Navigate to https://dao.rei.network/#/stakeforgas

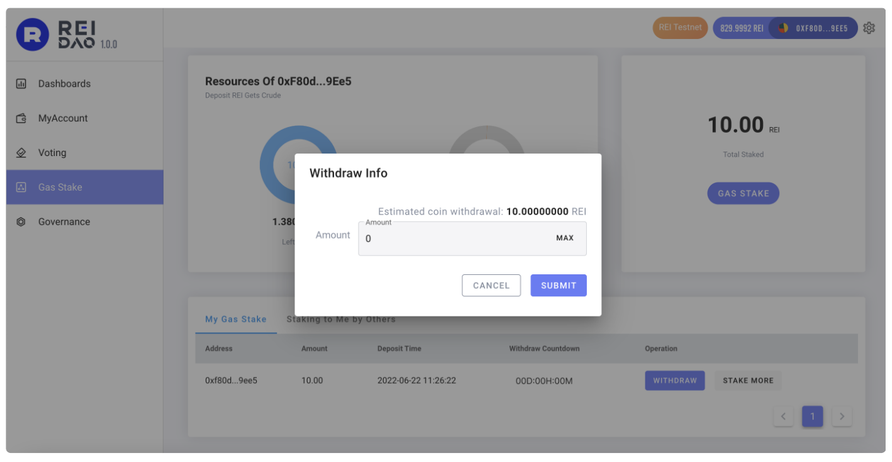
Step2: Click on `GAS STAKE` button, enter the `address` and `amount` to stake



Withdraw Gas Stakes

UI | Ethers

The staked `$REI` cannot be withdrawn within 3 days. After 3 days, `WITHDRAW` will be enabled.



About BLS Public key Registration

1. Download version 3.0.0 validator client

Create a new directory, then navigate into it, and check the current Node.js version to ensure that the node version is equal to or greater than v16.0.0.

```
node -v
```

If the Node.js version does not meet the requirement, it is recommended to use nvm (Node Version Manager) to switch to a different Node.js version.

```
nvm use 16
```

Once you have verified that the current Node.js version meets the requirements, proceed to execute the following command to install the 3.0.0 (pre-release) version of the client:

```
npm install @rei-network/cli@3.0.0
```

Eg:

```
mkdir rei-cli
cd rei-cli

node -v
v14.19.3

nvm use 16

node -v
v16.18.0

npm install @rei-network/cli@3.0.0
```

After the installation is completed, execute the following command to check the version number and confirm if the installation was successful:

```
npx rei --version
```

Eg:

```
[root@rei-testnet-validators rei-cli] npx rei --version
3.0.0
```

2. Generate bls keystore

Use the following command and enter the password according to the prompt to complete the generation of bls keystore.

```
npx rei bls new --datadir <path>
```

> **note:**
> - This command will create a new bls directory under the specified path, and generate a keystore file in the bls directory.
> - The specified path is recommended to be the original validator datadir (or migrate the generated bls directory to the original validator datadir), otherwise it will affect the unlocking of the bls private key after rei-dao hardfork.
> - Please keep the password safe, otherwise bls keystore will not be able to unlock the bls private key.

Eg:

```
[root@rei-testnet-validators rei-cli]  npx rei bls new --datadir ~/node
Your new bls secret keyfile is locked with a password. Please give a password. Do not
? Password: [hidden]
? Repeat password: [hidden]

Your new key was generated

Public key: 0xa91f44ef2da6a839fbad965461 5f017f98d2c5b189cd87e062fbec7c9188a4f951425c5e4
Path of the secret key file: /root/node/bls/UTC--2023-06-27T19-06-09.134Z--0xa91f44ef2d

If you have not registered the BLS public key, please go to https://dao.rei.network to
- You can share your publickey with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your block
- You must BACKUP your key file! Without the key, it's impossible to access block signat
- You must REMEMBER your password! Without the password, it's impossible to decrypt the
```

From the above information, we can know that the path of the bls keystore file generated in this example is:

```
/root/node/bls/UTC--2023-06-27T19-06-09.134Z--0xa91f44ef2da6a839fbad965461 5f017f98d2c5b
```

3. Obtain bls publicKey

First, let's enter the directory where the bls keystore file is located

```
cd ~/datadir/bls
```

Check the file list to get the name of the bls keystore file we generated

```
ls
```

Eg:

```
[root@rei-testnet-validators rei-cli] cd ~/node/bls
[root@rei-testnet-validators bls] ls
UTC--2023-06-27T19-06-09.134Z--0xa91f44ef2da6a839fbad965461 5f017f98d2c5b189cd87e062fbec
```

Check out the bls keystore file

```
cat UTC--2023-06-26T19-43-41.889Z--0x8dbfbd24c6e56fbe57cdb5f131e8b255ac7120be6a731b9150
```

The displayed data is as follows, where the publicKey is the bls public key we need

```
{
  "encryptedSecretKey": "J15160hb45d2dandcdd857a9cc289a1bfffe448b173 6Gccn6afJfac48020452:
  "iv": "2cff7f8e6eff32c40c6d8953b1030f391",
  "publicKey": "0x8dbfbd24c6e56fbe57cdb5f131e8b255ac7120be6a731b9150595 30d58fd0d4aba99e
}
```

4. Register the bls public key to the chain and check whether the registration is successful

Regarding the bls registration contract, its abi is as follows:

```
interface IValidatorBLS {

    function setBLSPublicKey(bytes calldata key) external;

    function getBLSPublicKey(address) external view returns (bytes memory);

    function isRegistered(address) external view returns (bool);

    function isBLSPublicKeyExist(bytes calldata) external view returns (bool);
}
```

Functions

setBLSPublicKey : register bls public key

Declaration

```
function setBLSPublicKey(bytes memory key) external
```

Args

Arg	Type	Description
key	bytes	bls public key

getBLSPublicKey : Query bls public key

Declaration

```
function getBLSPublicKey(address) external view returns (bytes memory)
```

Args

Arg	Type	Description
address	address	validator address

isRegistered : Query whether the address register bls public key

Declaration

```
function isRegistered(address) external view returns (bool)
```

Arg	Type	Description
address	address	validator address

isBLSPublicKeyExist : Query whether bls public key is registered

Declaration

```
function isBLSPublicKeyExist(bytes memory) external view returns (bool)
```

Arg	Type	Description
key	bytes	bls public key

For developers, you can call the setBLSPublicKey method of the validatorBLS contract to register the bls public key to the chain

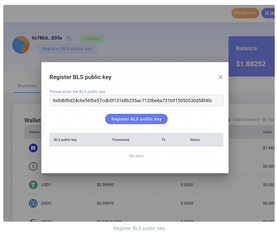
> **note:**
> - Before rei-dao fork, the contract address is `0x5b84072EE3e72a58A906FA15182beEdaB5298076` , after rei-dao fork, the contract address is `0x0000000000000000000000000000000000001009`
> - After the registration is complete, you can call the isRegistered method to check whether the registration is successful or call the getBLSPublicKey method to check whether the bls public key is consistent with the registered one

For general users, you can register bls public key to the chain through REI DAO
The steps are as follows:

UI

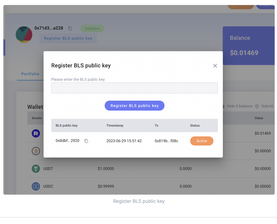
Step1: Navigate to https://dao.rei.network/#/myAccount/portfolio

Step2: Click on `Register BLS public key` button, enter the `BLS public key` to Register



Register BLS public key

Please note that it may take 1 to 2 minutes for the registration record to be updated here after registering with the BLS public key.



Register BLS public key