Technical description of the Waves Enterprise platform

Release master

https://wavesenterprise.com

Sep 06, 2022

BLOCKCHAIN-PLATFORM WAVES ENTERPRISE

CHAPTER

ONE

FEATURES OVERVIEW

The Waves Enterprise Blockchain Platform is a scalable digital infrastructure solution that combines the features of public and private blockchains for corporate and government use. The platform uses operation protocol, rather than business logic, to solve the problem of trust between parties. The *ProofofStake* (PoS), *ProofofAuthority* (PoA) and *Crash Fault Tolerance* (CFT) consensus mechanisms guarantee the correctness of data added to the blockchain, while decentralization provides counterparty independence for data access.

1.1 Waves Enterprise Blockchain Highlights

- Built on Scala programming language.
- Includes technologies and best use practices of use proven on the Waves public blockchain platform.
- Adapted for corporate and government use.
- Supports *PoS*, *PoA* and *CFT* consensus algorithms, and allows administrators to choose the most fitting one during deployment.
- Ensures high throughput rate.
- Supports Turingcomplete Docker *smart contracts*.
- Delivered as a set of microservices.
- Uses cryptographic algorithms certified by state regulators.
- Supports confidential and direct data exchange via private groups without loading data onto external networks.
- Implements the permission management system at the consensus level.
- Waves Enterprise web client features *transactions* explorer, wallet, creation of transactions, smart contract development, blockchain status monitoring, and permission management.

1.1.1 Waves Enterprise network deployment options

The Waves Enterprise platform offers three variants of blockchai deployment:

1. Operating in the main public network.
2. Operating in a private network anchored to the main network.
3. Operating in an independent private network.

1.2 Main network

The main network is supported by a consortium of companies from various economic sectors including banking, industrial, real estate, logistics, etc. Companies which use the main network may use public blockchain for their projects or for supplying blockchain processes, e.g. banking enterprises delivering fiat *gateways*, and state registrars granting access to cloudbased GOST cryptography.

1.3 Independent private network

Independent private networks may be used by companies that do not want to share their processes publicly. Waves Enterprise allows such companies to deploy a standalone private network out of the box and configure it in accordance with their business needs.

Following features are configurable:

- Consensus type.
- Cryptography provider.
- Number of nodes.
- Blockchain operating parameters.

1.4 Private network with block hashes broadcast to main network

This solution combines the advantages of public and private networks. Private networking allows companies to conceal private information from the public blockchain, while the broadcast of private block hashes to the main network ensures reliability of information, thanks to the scalability of the main network.

CHAPTER

TWO

OFFICIAL RESOURCES

- Official site of the blockchainplatform Waves Enterprise
- Official page on Github
- Official site of the blockchainplatform Waves

CHAPTER

THREE

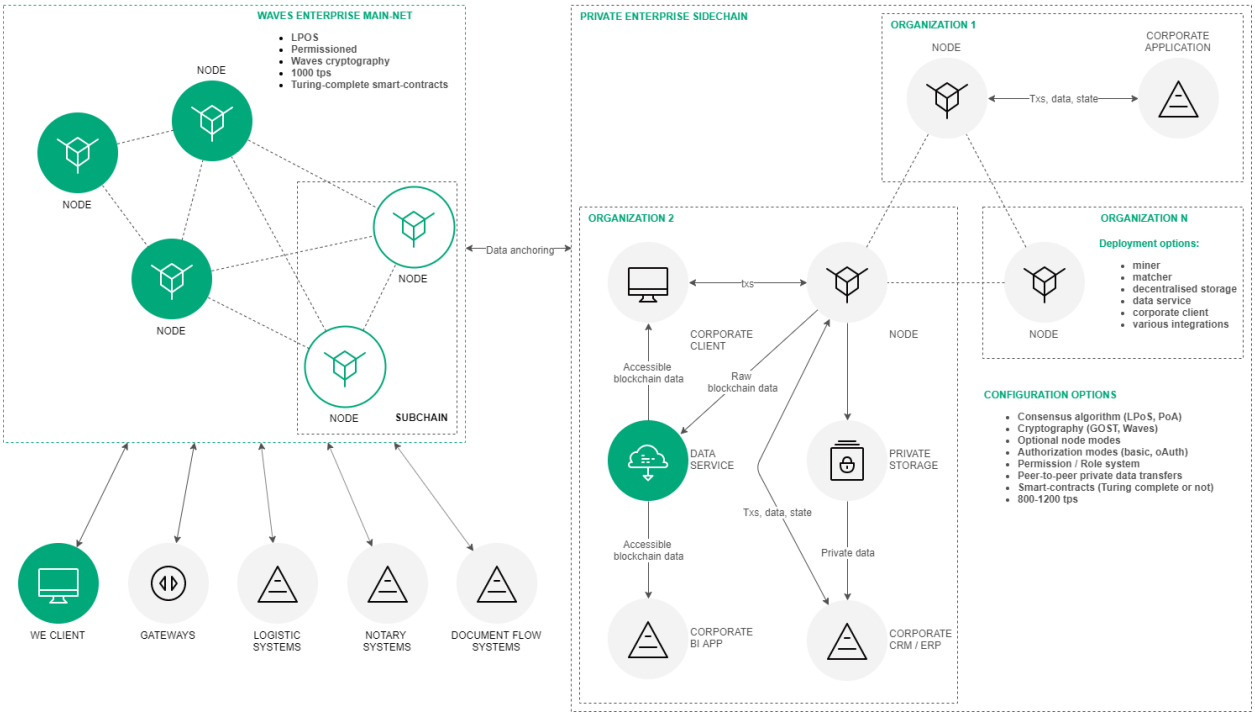
ARCHITECTURE

The Waves Enterprise platform is based on distributed ledger technology and represents a fractal network consisting of:

- A **master blockchain** (Waves Enterprise Mainnet) which secures the operation of the network serving as a global arbiter and a reference chain, and some multiuser networks;
- a number of separated **sidechains** that can be configured easily according to specific business needs.

This construction principle optimizes the platform for higher speeds, large volumes of calculations, consistency and availability of data, and resistance to malicious changes in information.

The *Anchoring mechanism* uses the strengths of both consensus algorithms to create a net configuration. For instance, the main Waves Enterprise blockchain is based on the *ProofofStake* consensus algorithm, which is supported by independent participants. At the same time, enterprise sidechains do not need to interact with miners and can use the *ProofofAuthority* algorithm. Sidechains are embedded in the main blockchain using the anchoring mechanism, placing cryptographic proof of transactions in the main blockchain network.



3.1 Node architecture and additional services

The node component is mandatory, since it ensures the functioning of and interaction within the blockchain network. Other components serve auxiliary purposes significantly simplifying user interaction with the blockchain platform. The Waves Enterprise Blockchain Platform instance consists of five basic modules and several additional microservices. The main modules include:

- **Node** The main software, which is installed on the computer and works directly with the blockchain.
- **Waves Enterprise corporate client** – A *webapplication* that provides contemporary and multifunctional user interface for the blockchain platform.
- **Smartcontracts module** – An environment for deploying and executing of Turingcomplete Docker smartcontracts. Docker containers with smartcontracts are deployed on remote virtual machine for additional security.
- **Data service** – A *service* that aggregates data from the blockchain in RDBMS (PostrgreSQL) storage and provides fulltext search on any information within the blockchain via the RESTful web service.
- **Приватное хранилище** компонент обеспечивают обработку и хранение приватных данных, а также коммуникации через зашифрованное соединение peertopeer. Приватное хранилище реализуется на базе БД PostgreSQL или S3 на Minio.

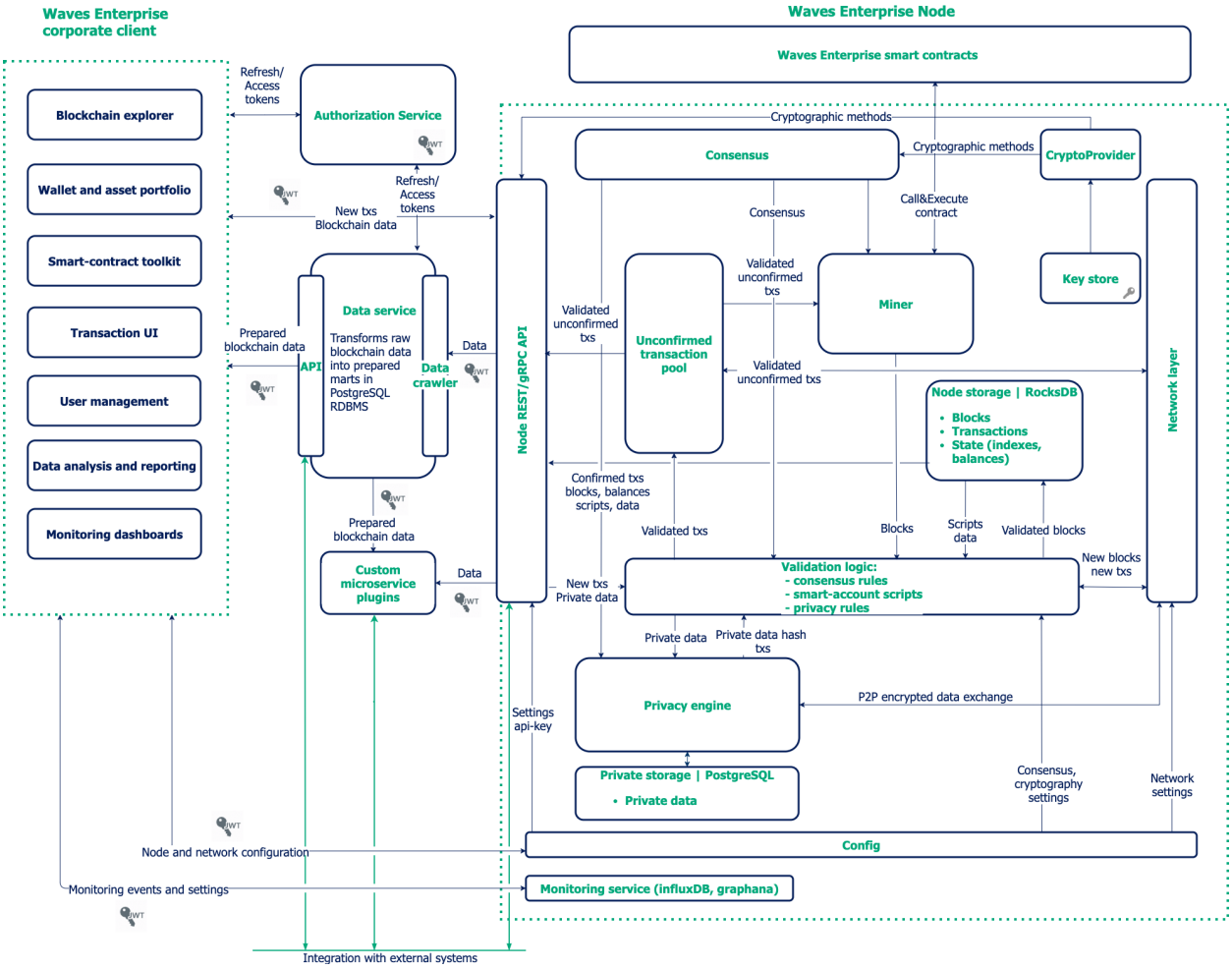
Additional services include:

- **Authorization service** – A single authorization service for system components.
- **Data crawler** A service that extracts data from blockchain node and loads it into dataservice component.
- **Generator** A service that generates key pairs for new accounts and creates `apikeyhash`.
- **Custom microservice plugins** A set of plugins for processing and customizing data transferred to and from external systems.
- **Monitoring Service** – An external monitoring service that uses an opensource database (InfluxDB) to store time rows with application data and metrics. The InfluxDB database is an opensource software and should be installed by the client separately.

Node components

The node includes the following internal components:

- **Node API** – A REST API and gRPC node interfaces which can receive data from the blockchain, sign and send transactions, send private data, and create and call smart contracts.
- **Node storage** – A system component that provides keyvalue storage (based on RockDB) for a full set of validated and confirmed transactions and blocks, same as the current state of objects.
- **Unconfirmed transaction pool (UTX pool)** – A component that provides a temporary storage and queue service for validated transactions until they are included into a block.
- **Consensus and cryptolibraries** – Configurable and customizable logical components responsible for achieving agreement between nodes and cryptographic algorithms.
- **Key store** A component used to store key pairs for the node itself and node users (optional). All keys are secured by passwords.
- **Miner** – A component responsible for creating transaction blocks that are recorded in the blockchain. The miner component is in charge of interaction with Dockersmart contracts.
- **Network layer** – A logic layer that provides interaction between nodes on the application level via network protocol over the TCP.


Waves Enterprise corporate client
Blockchain explorer
Wallet and asset portfolio
Smart-contract toolkit
Transaction UI
User management
Data analysis and reporting
Monitoring dashboards
Refresh/ Access tokens
Authorization Service
JWT
Refresh/ Access tokens
New txs Blockchain data
Data service
Transforms raw blockchain data into prepared marts in PostgreSQL RDBMS
API
Data crawler
Prepared blockchain data
Data
Prepared blockchain data
Custom microservice plugins
Data
Waves Enterprise Node
Waves Enterprise smart contracts
Cryptographic methods
Consensus
Cryptographic methods
CryptoProvider
Consensus
Call&Execute contract
Validated unconfirmed txs
Miner
Key store
Validated unconfirmed txs
Unconfirmed transaction pool
Validated unconfirmed txs
Node REST/gRPC API
Node storage | RocksDB
• Blocks
• Transactions
• State (indexes, balances)
Network layer
Confirmed txs blocks, balances scripts, data
Validated txs
Blocks
Scripts data
Validated blocks
New blocks new txs
Validation logic:
- consensus rules
- smart-account scripts
- privacy rules
New txs Private data
Private data
Private data hash txs
Privacy engine
P2P encrypted data exchange
Settings api-key
Private storage | PostgreSQL
• Private data
Consensus, cryptography settings
Network settings
Config
Node and network configuration
Monitoring service (influxDB, graphana)
Monitoring events and settings
Integration with external systems

- **Validation logic** – A logic layer containing such transaction verification rules as basic sign verification and advanced scripted verification.
- **Config** – A set of node configuration parameters specified in the `nodename.conf` file.
- **Monitoring Service** – An external monitoring service that uses an opensource database (InfluxDB) to store time rows with application data and metrics. The InfluxDB database is installed by the client separately.

CHAPTER

FOUR

WAVESNG PROTOCOL

The Waves Enterprise Operation Protocol provides performance advantages relative to other blockchains.

4.1 Terms

- **Block** — A set of transactions registered in the blockchain, signed by the miner, and containing a link to the proof of the previous block. Limited to 1 MB or 6000 transactions.
- **Round** — A period of time between the issuance of key blocks. This floating value is controlled by the consensus algorithm depending on the load on the network, averaging 40 seconds.
- **Proof of ownership** — The acquisition of mining rights in the PoS consensus.
- **Node** — A network host that runs the Waves Enterprise blockchain application.
- **Miner** — A node whose address has sufficient balance and a "mining" permission.
- **Key block** — A block that contains no transactions, only service information such as:
  - Miner public key — to verify proof of microblocks.
  - Amount of miner's fee for the previous block.
  - Miner's proof.
  - Link to previous key block.
- **Liquid Block** — A service term to describe the state of a block before issuing the next key block, i.e. completing its mining.
- **Microblock** — A service term for a set of transactions applied to the state of blockchain every 5 seconds. Limited to 500 transactions. Each microblock is signed by the miner's private key.

4.2 Protocol description

The WavesNG protocol — a protocol developed by Waves Platform based on BitcoinNG to increase the throughput of the Waves blockchain based on the architecture on which Waves Enterprise is implemented. The idea of the protocol is to create microblocks continuously, rather than create one large block in each round of mining. Small blocks can be forwarded and checked more quickly.

Mining rounds begin with the release of a key block. The time of appearance of each keyblock and the address of the miner specified in it are determined by a *consensus*. The key block contains no transactions and is quickly generated. Then, until the next block appears, microblocks with transactions without proof of share are generated every 5 seconds, which also increases the processing speed. Each microblock references

the previous one. The key block is added to the blockchain as soon as the next miner releases its key block with a link to it.

This approach reduces the time to confirm a transaction compared to other blockchains.

4.2.1 1. Process for Creating a Liquid Block

1. The mining address is determined by consensus.
2. A miner creates and distributes a key block on the network.
3. Every 5 seconds, the miner creates a microblock containing transactions and sends it out to the network. Each microblock must be linked to the previous microblock or key block.
4. The process continues until a new valid key block appears on the network.

4.2.2 2. Miner reward mechanism in WavesNG

The Waves Enterprise protocol offers financial incentive for participants to comply with the rules of the blockchain. 40 % of the block transaction fee is distributed to the miner who created the block, and 60 % of the fee is given to the miner of the following block. The fee credit transaction is performed after 100 blocks to ensure a trust interval of checks.

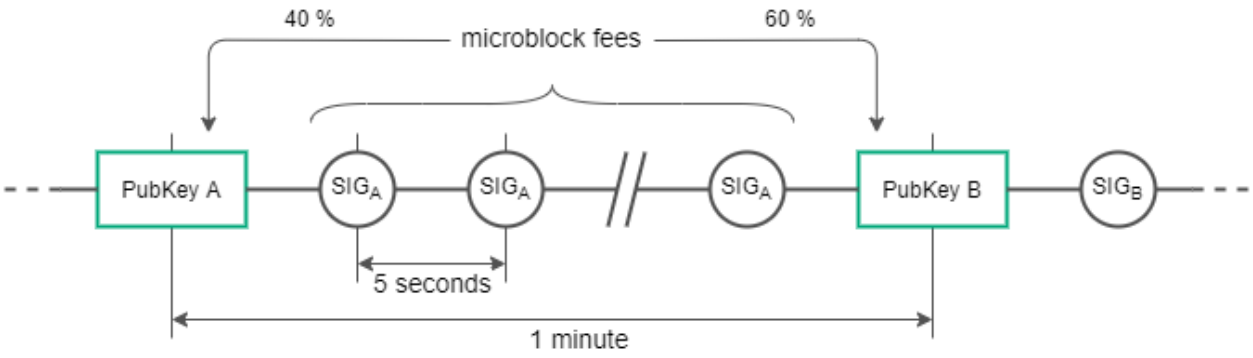


Fig. 1: Fee distribution diagram

4.2.3 3. Conflict resolution

A miner that continues the chain by creating two microblocks with the same parent is punished by deprivation of income from fees; the discoverer of the fraud receives the miner's award for the block. The distributed nature of blockchain means each node stores a copy of the blockchain. When the next microblock appears, the node applies changes to its copy of the blockchain and checks it against other nodes of the network. At this point, inconsistencies in transactions can be detected.

CHAPTER

FIVE

CONSENSUS ALGORITHMS

Blockchain is a decentralized system with no central authority. This makes the system noncorrupt, but it also creates difficulties with final decisionmaking and organization of work. These problems are solved by a consensus mechanism, which allows the blockchain's participants to reach agreement. Voting takes into account the majority opinion without the interests of the minority, but it also guarantees an agreement that benefits the entire network.

You can choose the consensus mechanism during the initial configuration of the network. The description of available mechanisms, as well as their pros and cons, are described below.

5.1 Алгоритм консенсуса PoS / LPoS

Proof of ownership with the right to lease. In PoS systems, the creation of a block does not require energy-intensive calculations, the miner's task is to create a digital block proof.

5.1.1 Proof of Stake

The mechanism for allocating block creation rights is based on the number of tokens in the user's account. The more tokens a user has, the more likely he or she can create a block.

In Proof of Stake consensus the right to generate a block is determined by pseudorandom way, because by knowing the previous miner and balances of all users in the system the following miner can be identified. This is possible due to a deterministic computation of a block's generating signature, which can be obtained by SHA256 hashing of current block's generating signature and the account's public key. The first 8 bytes of the resulting hash is converted to a number, referred to as the account hit X_n and will be a pointer to the following miner. The time of block generation for account i is calculated as:

$$T_i = T_{min} + C_1 \log(1 - C_2 \frac{\log \frac{X_n}{X_{max}}}{b_i A_n})$$

where:

- b_i a stake (stake of participant's balance of overall balance of the system)
- A_n baseTarget, the adaptive ratio, regulating the average time of issue of the block;
- X_n an account hit;
- T_{min} 5 seconds, a constant defining the minimum time interval between blocks;
- C_1 a constant, which equals 70 and adjusts the form of allocation of the interval between blocks;
- C_2 a constant which equals 5E17 and adjusts the baseTarget value (complexity).

Based on this formula, the probability of selecting the participant to be rewarded depends on the participant's stake of assets in the system. The bigger the stake, the higher the chance of reward. The minimum number of tokens needed for mining is **50000 WEST**. BaseTarget is a parameter that maintains the block generation time within a given range. BaseTarget in its turn is calculated as:

$$(S > R_{max} \rightarrow T_b = T_p + max(1, \frac{T_p}{100})) \wedge (S < R_{min} \wedge \wedge T_b > 1 \rightarrow T_b = T_p - max(1, \frac{T_p}{100}))$$

where

- $R_{max} = 90$ a maximum reduction of complexity when the block generation time in the network exceeds 40 seconds;
- $R_{min} = 30$ a minimal increase of complexity when the block generation time in the network is less than 40 seconds;
- S the average generation time, at least for the last three blocks;
- T_p the previous baseTarget value;
- T_b the computed baseTarget value.

For an advanced description of technical features and enhancements of the classic PoS algorithm, see this article.

Advantages Over Proof of Work

The absence of complex calculations allows PoS networks to lower the hardware requirements for system participants, which reduces the cost of deploying private networks. No additional emission is required, which in PoW systems is used for rewarding miners for finding a new block. In PoS systems, a miner receives a reward in the form of fees for transactions which appeared in its block.

5.1.2 Leased Proof of Stake

A user who has an insufficient stake for effective mining may transfer his balance for lease to another participant and receive a portion of the income from mining. Leasing is a completely safe operation, as tokens do not leave the user's wallet, but are delegated to another miner, which gives the miner a greater opportunity to earn mining rewards.

5.2 PoA consensus algorithm

In a private blockchain, tokens are not always needed. For example, a blockchain can be used to store hashes of documents exchanged by organizations. In this case, in the absence of tokens and fees from transactions, a solution based on the PoS consensus algorithm is redundant. The Waves Enterprise Blockchain Platform offers the option of a Proof of Authority (PoA) consensus algorithm. Mining permission is issued centrally in the PoA algorithm, which simplifies the decisionmaking compared to the PoS algorithm. The PoA model is based on a limited number of block validators, which makes it scalable. Blocks and transactions are verified by preapproved participants who act as moderators of the system.

5.2.1 Algorithm description

An algorithm determining the miner of the current block is formed based on the parameters below. The parameters of the consensus are specified in the `consensus` block of the node configuration file.

- t the duration of a round in seconds (the parameter of the node configuration file: roundduration).
- t_s the duration of a synchronization period, calculated as t*0.1, but not more than 30 seconds (the parameter of the node configuration file: syncduration).
- N_{ban} a number of missed consecutive rounds for issuing the ban for the miner (the parameter of the node configuration file: warningsforban);
- P_{ban} a share of the maximum number of banned miners, in percentage from 0 to 100 (the parameter of the node configuration file: maxbanspercentage);
- t_{ban} the duration of the miner ban in blocks (the parameter of the node configuration file: banduration-blocks).
- T_0 the unix time for generation the Genesis block.
- T_H the unix time for generation of H Block, a key block for NG.
- r the round number, calculated as ($T_{Current}$ T_0) div (t+ t_s).
- A_r the leader of round r, which has the right to create key blocks and microblocks for NG in the round r.
- H the height of the chain in which the key block and microblocks for NG are created. The leader of round A_r has the right to generate a block at height H.
- M_H the miner issuing block at height H.
- Q_H the queue of miners active at height H.

The Q_H queue is generated using addresses which are given mining permissions by a permission transaction, which was not revoked until height H and did not expire until the time T_H .

The queue is sorted by the time stamp of the mining rights transaction. The node which was granted the rights earlier will be higher in the queue. To keep the network consistent, this queue will be the same on each node.

A new block is created at each round r. A round lasts t seconds. After each round, t_s seconds count down to complete data synchronization in the network. During the synchronization period, microblocks and key blocks are not generated. For each round, a single leader, A_r , has the right to create a block in this round. A leader can be defined on each node of the network with the same result. The leader of the round is defined as follows:

1. Miner M_{H1} is defined, which created the previous key block at height H1.
2. The Q_H queue of active miners is calculated.
3. Inactive miners are excluded from the queue (see more in *Exclusion of inactive miners*).
4. If the H1 block miner (M_{H1}) is in the Q_H queue, the following miner becomes the leader A_r .
5. If the H1 block miner (M_{H1}) is not in the Q_H queue the miner following the H2 block miner (M_{H2}) becomes the leader A_r and so on.
6. If no miners of blocks (H1..1) are in the queue, the first miner in the queue becomes the leader.

This algorithm identifies and checks the miner, which creates each block of the chain by calculating the list of authorized miners for each moment of time. If the block was not created by the designated leader within the allotted time, no blocks are generated within that round, and the round is skipped. Leaders who skip block

generation are temporarily excluded from the queue by the algorithm described in the paragraph *Exclusion of inactive miners*.

The block generated by the leader A_r with the time of the block T_H from the halfinterval (T_0 +(r1)*(t+t_s); T_0 +(r1)*(t+t_s)+t] is determined to be valid. The block created by the miner out of its turn or not in time is considered invalid. After a round of t duration, the network synchronizes the data for t_s . The leader A_r has t_s seconds to propagate the validation block over the network. If any node of the network during t_s has not received a block from the leader A_r , this node recognizes the round as "skipped" and expects a new H block in the next round r+1, from the following leader A_{r+1} .

Several consensus parameters — type (PoS or PoA), t, t $_s$— are specified in the configuration file of the host network. *The parameter T should be the same for all network participants*, otherwise the network will fork.

5.2.2 Synchronization of time between network hosts

Each host should synchronize the application time with a trusted NTP server at the beginning of each round. The server address and port are specified in the node configuration file. The server must be available to each network node.

5.2.3 Exclusion of inactive miners

If any miner has missed the block creation N_{ban} times in a row, this miner is excluded from the queue at t_{ban} subsequent blocks, which is determined by (`bandurationblocks` parameter in the configuration file). The exception is made by each node on its own based on the calculated queue Q_H and information about block H and miner M_H . The P_{ban} parameter specifies the maximum allowable share of excluded miners in the network relative to all active miners at any given time. If at achievement of N_{ban} round passes, the maximum share of the excluded miners P_{ban} is reached, the exception of the next miner is not made.

5.2.4 Monitoring

The PoA consensus monitoring helps to identify how nonvalid blocks are created and distributed, as well as how miners skip the queue. Network administrators perform additional troubleshooting and blocking of malicious nodes.

To monitor the process of generating blocks using the PoA algorithm, the following details are entered in InfluxDB:

- Active list of miners sorted by granting of mining rights.
- Scheduled round timestamp.
- Actual round timestamp.
- Current miner.

5.2.5 Changing consensus settings

Changing consensus parameters (time of round and synchronization period) is performed based on the node configuration file (see the insert) at the height fromheight. If a node fails to specify new parameters, the transaction will fork.

Sample configuration:

```
// specifying inside of the blockchain parameter
consensus {
  type = poa
  sync-duration = 10s
  round-duration = 60s
  ban-duration-blocks = 100
  changes = [
    {
      from-height = 18345
      sync-duration = 5s
      round-duration = 60s
    },
    {
      from-height = 25000
      sync-duration = 10s
      round-duration = 30s
    }]
 }
```

5.3 CFT consensus algorithm

When information is exchanged extensively in a corporate blockchain, consistency between the network elements that form a single blockchain is important. And the more participants are in the exchange, the more likely it is that an error will occur: a hardware failure by one of the participants, network problems, and so on. This can lead to *forks* of the main blockchain and, as a consequence, *rollback* of a block that seems to be already formed and included in the blockchain. In this case, the blocks subject to the rollback begin to be mined again and become unavailable in the blockchain for some time. This, in turn, can affect the business processes that use the blockchain. The CFT (Crash Fault Tolerance) consensus algorithm is designed to prevent such situations.

5.3.1 Algorithm description

The CFT consensus algorithm is based on the *PoA* with an added phase for voting of **mining round validators**: network participants that are automatically appointed by the consensus algorithm. This approach guarantees the following:

- more than a half of participants (validators) are familiar with a definite block and have validated it;
- the block will not be rollbacked and will be published in the blockchain;
- there will be no parallel chain in the network.

This is achieved by the finalization of a produced block. The finalization itself is based on the consensus of majority of round validators (50% + 1 vote). In accordance with this consensus, the decision of block broadcasting is taken. If this majority has not been achieved, mining will be stopped up to restoring of network cohesion.

Like PoA, CFT depends on a current time, starting and ending time of each mining round is calculated upon the basis of a *genesis block* timestamp. Basic parameters that form an algorithm that is used for appointment of a current block miner are also identical to the PoA parameters (see the section Algorithm description). For validation of blocks, the `consensus` block of the node configuration file has been expanded with two new parameters:

- **maxvalidators** – limit of validators participating in a current round;
- **finalizationtimeout** – time period, during which a miner waits for finalization of the last block in a blockchain. After that time, the miner will return the transactions back to the UTX pool and start mining the round again.

The following terms are used for the following description of CFT functionality:

- t round duration in seconds (parameter of the node configuration file: roundduration).
- t_{start} round start time.
- t_{sync} blockchain synchronization time (t_{start} + t).
- t_{end} round end time.
- t_{fin} time period during which a miner waits for finalization of the last block (parameter of the node configuration file: finalizationtimeout).
- V_{max} limit of validators taking part in voting (parameter of the node configuration file: maxvalidators).

5.3.2 Voting

Voting is performed in each round, nodes with the miner role can take part in it. Voting starts upon t_{sync} and ends by t_{end} + t_{fin} . Within each time period defined for voting, *voting of validators* and *voting of current round miner* are performed. Each validator of a round can send multiple votes, but a miner can vote only once for its last microblock.

For voting, instance of a vote is used, which includes following parameters:

- **senderPublicKey** – a public key of a validator which has formed a vote;
- **blockVotingHash** – hash of a *liquid block* with votes confirmed by a validator;
- **signature** – vote signature formed by a validator.

Defining of round validators and their voting

In order to define validators that can vote in a current round, a configurable node parameter maxvalidators (V_{max}) is used. If the number of active miners minus the current round miner does not exceed V_{max} , each of them can take part in voting. Otherwise, in order to define validators of a current round, the pseudorandom selection algorithm is used which allows to exclude the influence of a particular miner on choices of voters.

Voting of validators start under two preconditions:

- the next attempt to vote falls within the time interval required for voting;
- the address of the current node is one of the defined validators of the round for voting.

After the end of the round validators voting, the miner voting is started.

Voting of current round miners

The miner's vote is triggered under two conditions:

- the next attempt to vote falls within the time interval required for voting;
- the address of the current node is the miner of the round.

A vote is considered valid if it was issued by an address that is in a list of validators of the current round and has a correct signature. As soon as a miner gains the required number of votes, voting time slot is checked. Then the finalizing microblock with all votes is released. The block with votes is considered finalized.

5.3.3 Mining features

The basic rules of CFT consensus mining are identical to the PoA consensus rules. However, an additional mechanism has been introduced to ensure consensus fault tolerance.

With CFT consensus, another mining attempt is considered a failure in case the last received block has not been finalized in other words, a microblock with valid votes has not been applied to the state. In this case, if the mining attempts exceed the $t_{start} + t_{fin}$, the node decides to return all transactions from the last block back to the UTX pool, after that the round starts mining again.

In order to exclude returning of your transactions into the UTX pool, it is highly recommended to work not with the current (liquid) blockm but with a finalized one that has been already validated by the network participants.

5.3.4 Selecting a channel for synchronization

The PoS and PoA consensus algorithms use a module that selects the strongest chain for synchronization by comparing the data of the involved nodes. CFT uses a different selection mechanism, which also increases system fault tolerance: it selects a random channel from the channels that are active at the moment of synchronization. The list of active channels is constantly updated during the system operation, and the synchronization time with a particular channel is limited to evenly distribute the load of the network.

5.3.5 Changing consensus parameters

Like in the PoS and PoA consensus algorithms, the consensus parameters are configured in the node configuration file. The configuration example is stated below:

```
// specifying inside of the blockchain parameter
consensus {
  type = cft
  sync-duration = 10s
  round-duration = 60s
  ban-duration-blocks = 100
  max-validators = 16
  finalization-timeout = 5s
 }
```

CHAPTER

SIX

CRYPTOGRAPHY

The Waves Enterprise platform provides the possibility to choose the cryptography used depending on the specifics of the project under implementation and the jurisdiction of the customer.

6.1 Hashing

Hashing operations in the platform are performed by Blake2b256 and Keccak256 functions sequentially, or by "Stribog" function in accordance with GOST R 34.112012 "Information Technology. Cryptographic protection of information. Hash function". The output data block size is 256 bits.

6.2 Electronic signature

Algorithms for key generation, formation and verification of electronic signature are implemented on the basis of Curve25519 elliptic curve (ED25519 with X25519 keys), or in accordance with GOST R 34.102012 "Information technology. Cryptographic protection of information. The processes of formation and verification of electronic digital signature".

6.3 Data encryption

The platform implements the ability to encrypt data using session keys based on the DiffiHelman protocol. This operation is used to encrypt any type of text information, such as smart contract data, which should not be available to other blockchain participants. Encryption can be performed individually for each recipient, with the formation of a unique instance of ciphertext, or with the formation of a single ciphertext for a group of recipients.

The algorithms used for symmetric encryption comply with the AES standard or GOST R 34.122015 "Information technology. Cryptographic protection of information. Block cipher".

Symmetric CEK and KEK keys are used to encrypt/decrypt data. CEK (Content Encryption Key) is the key for the encrypting text data, KEK (Key Encryption Key) is the key for encrypting the CEK. The CEK key is generated by a node randomly using the appropriate hashing algorithms. The KEK key is generated by a node based on DiffieHellman algorithm, using public and private keys of sender and recipients, and is used to encrypt the CEK key.

For a description of encryption methods and their use, see *Data encryption operations*.

Important: An encryption algorithm of the offchain protocol for private data transfer depends on a used version of the node. For instance, in an actual version 1.5, in case of GOST encryption usage, the protocol establishes an encrypted TLSlike connection with the use of the Kuznechik encryption algorhithm.

CHAPTER

SEVEN

MANAGING PERMISSIONS

The Waves Enterprise blockchain platform implements a closed (permissioned) blockchain model that can only be available only for participants *authorized* by the administrator.

The platform also has a role model, where each role gives the participant specified permissions. For more information about roles in the Waves Enterprise blockchain network, see the next section.

7.1 Roles description

permissioner

`Permissioner` participant is the network administrator and has the right to assign or delete any roles of network participants. As a rule, the `permissioner` role is assigned to participants when starting the blockchain network.

sender

`Sender` participant has the right to send transaction into the blockchain network.

> **Attention:** The `sender` role is defined in a genesis block, that is why it is unavailable in networks created with the use of a platform version earlier than 1.5.0. The example of a `sender` role configuration in a node configuration file is stated in the section *Node configuration file*.

blacklister

`Blacklister` participant has the right to send assign or delete the `banned` role to other participants.

miner

`Miner` participant has the right to create blocks.

issuer

`Issuer` participant has the right to issue, reissue, and burn tokens.

contract_developer

`Contract_developer` participant has the right to install (to deploy) in the blockchain. For more information about smart contracts, see the section *Docker smart contracts*.

connectionmanager

`Connectionmanager` has the right to connect and disconnect blockchain nodes. For more information about new nodes connection, see the section *Access managing*.

banned

The `banned` role is given to that nodes, which are temporarily or permanently restricted in their actions in the blockchain network.

7.2 Update the permission list

Only a `permissioner` node can change the list of permissions. To add or remove roles use the *102 Permission Transaction*. When changing the list of permissions, the node performs the following checks:

1. *102* transaction sender is not in the **blacklist**.
2. Sender has the `permissioner` role.
3. The `permissioner` role is currently active for the transaction sender.
4. The role specified in the *102* transaction is inactive if it is added to the address, and active if it is removed from the address.

To place a selected node in the **blacklist**, a `permissioner` node assigns the `banned` role to the selected address by sending the *102* transaction to the blockchain with the appropriate parameters.

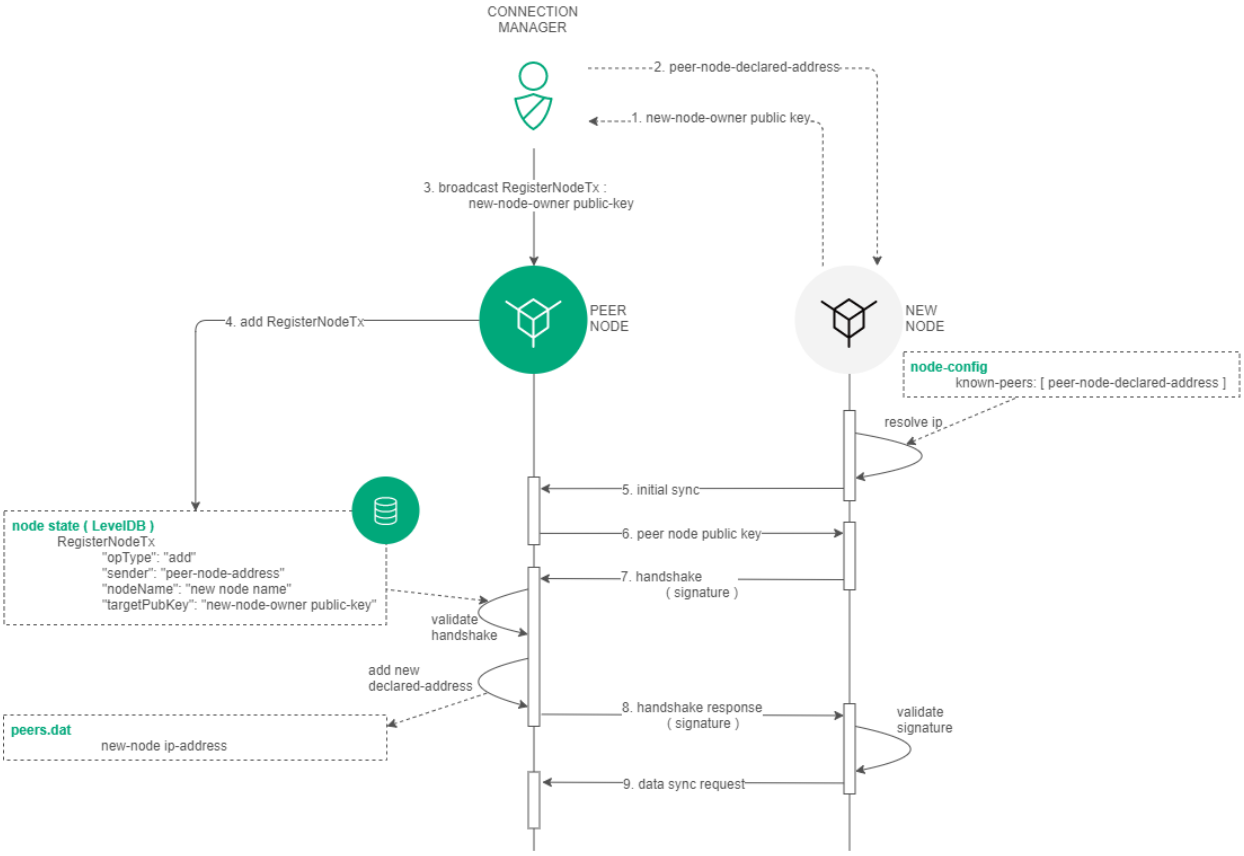
To assign any other roles (miner, contract developer, tokens managing node), a `permissioner` node issues a *102* transaction with the appropriate parameters. After the transaction enters the blockchain, changes to the permissions of the selected nodes will be completed.

7.3 Blockchain access managing

Only a user with the "Connection Manager" role can add new participants to the Waves Enterprise blockchain. The *111 RegisterNode* transaction is used to connect a new node to the network. This transaction contains the credentials of the connected node. Each node creates and updates the table, which includes all approved network participants.

A *handshakemessage* accompanies each connection attempt. This message specifies service information and proof that the connecting user belongs to the connected network. More simply, it is a set of public keys with the electronic signature of the participant. Since the public key of the connected participant is already stored in the blockchain, the participant who received the handshake request can verify the signature and the public key within the blockchain. If the verification is successful, the participant generates a response to the handshake request, and the connection between parties is established. After successful connection, participants perform network synchronization as well as synchronization of the blockchain and network addresses of nodes, which is necessary in the process of sending *private data*.

The process of disconnecting a participant from the network is similar to the process of connection, except that the "Connection Manager" user sends the *111 RegisterNode* transaction with the `"opType": "remove"` parameter. Since the handshake request is executed once every 30 seconds, the next request after the participant is removed from the network will be denied, as the connected participant would now lack credentials in the blockchain node table.


CONNECTION MANAGER
2. peer-node-declared-address
1. new-node-owner public key
3. broadcast RegisterNodeTx : new-node-owner public-key
4. add RegisterNodeTx
PEER NODE
NEW NODE
node-config
known-peers: [peer-node-declared-address]
resolve ip
5. initial sync
6. peer node public key
7. handshake (signature)
node state (LevelDB)
RegisterNodeTx
"opType": "add"
"sender": "peer-node-address"
"nodeName": "new node name"
"targetPubKey": "new-node-owner public-key"
validate handshake
add new declared-address
8. handshake response (signature)
validate signature
peers.dat
new-node ip-address
9. data sync request

CHAPTER

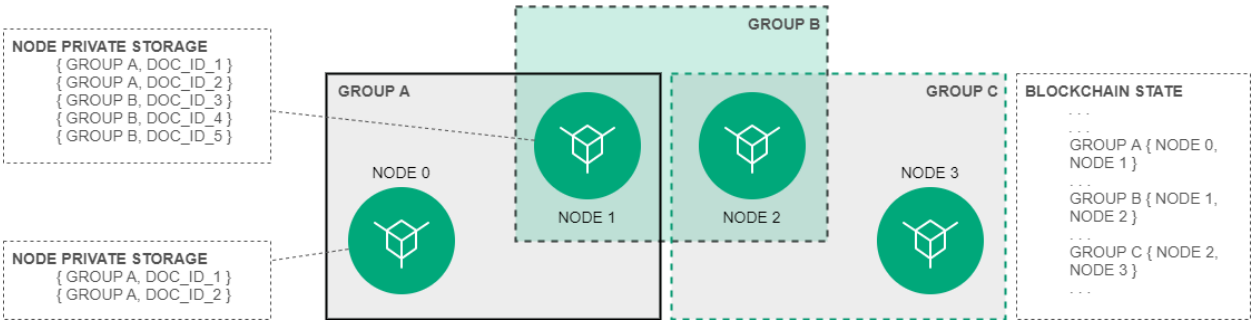
EIGHT

DATA PRIVACY

The Waves Enterprise Blockchain Platform provides confidential data transfer and storage between participants interacting on the network. The protection of confidential data during its transfer and storage is provided by a set of groups, which contain a list of participants that can interact with private data.

The Waves Enterprise platform supports two options for storing confidential data:

- PostgreSQL
- S3 based on Minio servers



8.1 Access groups

Access groups are created by network participants who need to arrange a private data exchange. Any participant can create an access group and add into it any number of other participants. Only nodes can exchange information within a group.

The group contains the following parameters:

- name (policyName);
- description (Description);
- the list of confidential data recipients (Recipients);
- the list of the policy owners with editing rights (Owners).

The access group is created by sending a *CreatePolicy* transaction (type = 112, group creation) to the blockchain.

Owners can change the access group by sending the *UpdatePolicy* transaction (type = 113, group editing) to the blockchain.

For external access and getting the information about groups there are using specified *API Node* requests: `GET /privacy/{policy}/recipients`, `GET /privacy/{policy}/getHashes`, `GET /privacy/getInfo/{hash}`.

8.2 Sending and receiving the data

The data is sent via `POST /privacy/sendData` request through its own node of the organization, which checks whether the sender is a member of the specified group. If that check is successful, the data is written to the node store, and the *PolicyDataHash* transaction (type = 114, sending the data hash) is initiated with the calculated hash sum of the data. The size limit for transferring data to the network is 20 MB.

When a receiving party receives a transaction with the hash sum from the transmitted data, it checks whether the blockchain node is involved in the group specified in the transaction. If the participant belongs to the group, the `getPrivateData` request for confidential data is executed at the network address of the group participant via P2P connection. To ensure the security of data transmission over an unprotected communication channel the DiffeyHellman cryptographic protocol is used.

CHAPTER

NINE

ACTIVATION OF BLOCKCHAIN FEATURES

The Waves Enterprise blockchain platform supports an opportunity to activate blockchain features through voting of nodes: in other words, the **soft fork mechanism**. Activation of new features is an irreversible action, because the blockchain does not support rollback of soft forks.

Only nodes with the `miner` role can take part in the voting, because votes of each node are attached to a block created by this node.

9.1 Voting parameters

Identifiers of features supported by a node are stated in the `supported` string of the `features` block in the `node` section of the node configuration file.

```
features {
  supported = [100]
}
```

Voting parameters are defined in the `functionality` block of the node configuration file:

- `featurecheckblocksperiod` voting period (in blocks);
- `blocksforfeatureactivation` number of blocks with a feature identifier required for activation of this feature.

By default, each node is set in a way that it votes for all supported features.

Attention: Voting parameters of a node cannot be changed during blockchain operation: these parameters should be unified for the entire network in order to provide full synchronization of nodes.

9.2 Voting procedure

1. During a mining round, a miner node votes for features included in the `features.supported` block, if they have not been activated in the blockchain before: feature identifiers are put into the `features` field of each block during its creation. After that, created blocks are published in the blockchain. So, all nodes with the `miner` role vote for their features during the `featurecheckblocksperiod`.
2. After the `featurecheckblocksperiod`, the system performs counting of votes identifiers of each feature in created blocks.
3. If a voted feature collects a number of votes that is greater or equal to the `blocksforfeatureactivation` it gets an **APPROVED** status.

4. The approved feature is activated after the `featurecheckblocksperiod` interval starting from a current blockchain height.

9.3 Usage of activated features

When activated, a new feature can be used by all blockchain nodes that support it. If any node does not support an activated feature, it will be disconnected from the blockchain in a moment of a first transaction using this unsupported feature.

When a new node is connected to the blockchain, it will automatically activate all previously voted and activated features. Activation is performed during synchronization of the node, if the node itself supports activated features.

9.4 Preliminary activation of features

All features available for voting can be also forcibly activated while starting a new blockchain. This can be set in the `preactivatedfeatures` block of the `blockchain` section in the node configuration fail:

```
pre-activated-features = {
  ...
  101 = 0
  }
```

Blockchain height for activation of a certain feature is stated after an equal mark in front of every feature.

9.5 List of available feature identifiers

Identifier	Description
100	Activation of the LPoS consensus algorithm
101	Support of gRPC by Docker smart contracts
119	Optimization of performance for the PoA consensus algorithm
120	Support of sponsored fees
130	Optimization of performance for miner ban history
140	Support of atomic transactions
160	Support of parallel creation of liquid blocks and microblocks

CHAPTER

TEN

ATOMIC TRANSACTIONS

Waves Enterprise platform supports the execution of atomic operations. Such operations consist of several actions, and are either performed completely or not performed at all. To do this, the system has *120* transaction which is a container that holds two or more signed transactions.

The *120* transaction supports the following transaction types:

- *4 Assets transfer*, version 3
- *102 Adding / removing permissions*, version 2
- *103 Contract creation*, version 3
- *104 Contract calling*, version 4
- *105 Contract execution*, versions 1 and 2
- *106 Contract disabling*, version 3
- *107 Contract updating*, version 3
- *112 Privacy group creation*, version 3
- *113 Privacy group updating*, version 3
- *114 Privacy data edding*, version 3

The key difference between new versions of transactions that are supported by the *120* atomic transaction is the presence of the *atomicBadge* tag field. This field contains the trusted address of the transaction sender *trustedSenders* to add to the transaction container *120*. If the sender's address is not specified, then the sender is the address from which the *120* transaction is sent to the blockchain.

10.1 Processing an atomic transaction

The *120* transaction has two signatures. The first transaction is signed by the sender to be successfully sent to the network. The second signature is generated by a miner and is necessary for adding a transaction to the blockchain. When adding *120* a transaction to the unconfirmed transaction pool, its signature is checked, as well as the signatures of all transactions included in the container. Validation of such transactions is performed according to the following rules:

- The number of transactions should be more than one.
- All transactions should have different IDs.
- The list of transactions should contain only supported types of transactions. To put a single atomic transaction to another is not allowed.

There should be no executed transactions inside an atomic transaction sent to the UTX pool, and the `miner` field should be empty. Also there should be no executable transactions inside an atomic transaction that is included in a block, and the `miner` field should not be empty.

After executing an atomic transaction, the block gets its "copy" formed according to the following rules:

- The `miner` field does not participate in the formation of the transaction signature and is filled in with the public key of the block miner.
- The block miner generates an array of `proofs`, the source of which is the IDs of transactions included in an atomic transaction. When included in a block, an atomic transaction has 2 signatures – the signature of the original transaction and the signature of the miner.
- If the list contains executable transactions, they are replaced with executed transactions. When validating an atomic transaction as part of a block, both signatures are checked.

10.2 Creating the atomic transaction

To create an atomic transaction, you need access to the node *REST API*.

1. The user selects from the list of supported transactions those transactions that should be performed as an atomic operation.
2. Correctly fills the fields of all transactions and signs them using the *sign* method.
3. Next, the user fills the field `transactions` of the *120* transaction with the data of signed but not sent to the blockchain transactions.
4. After entering all the transaction data, the user signs and sends the *120* transaction to the blockchain.

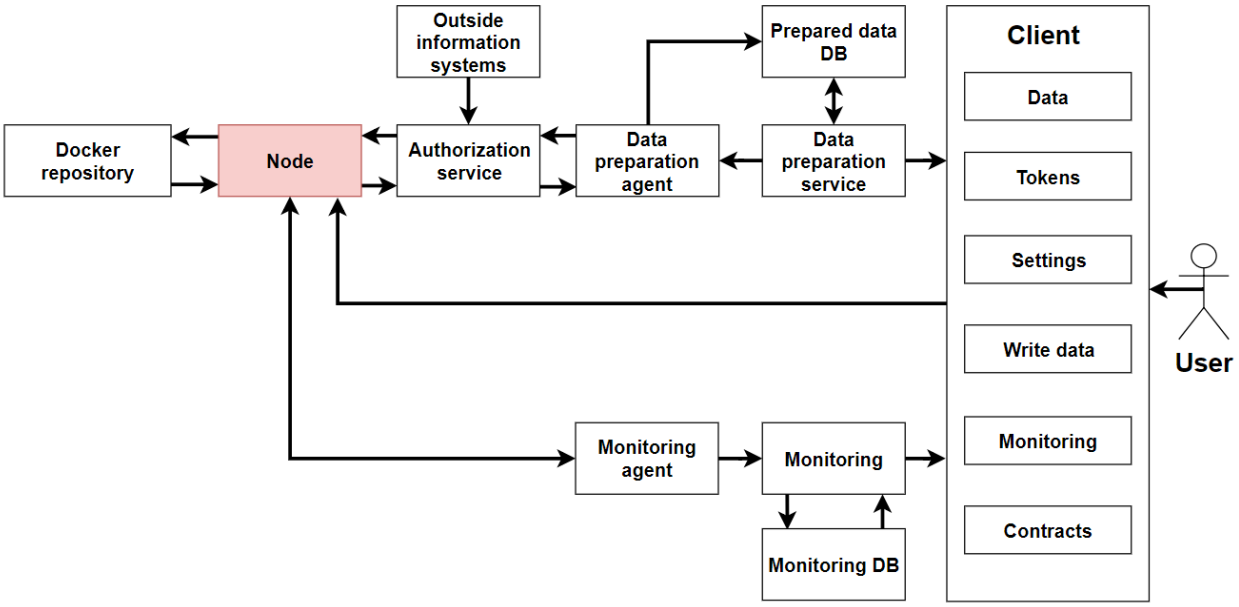
Attention: If you are creating an atomic transaction that includes *114* transactions, set the `broadcast = false` parameter when signing it.

CHAPTER

ELEVEN

CLIENT

Waves Enterprise client is a convenient way to manage Waves Enterprise blockchain. Client is intended for operations in the Waves Enterprise *public network*.



The client includes sections for use of all blockchain features:

- **Network stats** the page contains general information about the network and various statistics.
- **Explorer** — allows to find information about transactions or users through flexible search and advanced filter system.
- **Tokens** — allows to transfer, issue, lease tokens.
- **Contracts** — provides tools for publishing and calling docker contracts. Contracts are available for publishing from the repository, the address of which was specified when the client was built.
- **Data transfer** allows you to send data transactions and files from the interface, as well as work with privacy groups to transfer confidential data.
- **Network settings** allows you to view information about nodes in the network and calculate the leasing payouts.
- **Write to us** contact form for Waves Enterprise technical support. You can leave a free comment, and it will be reviewed by technical support.

You can find your profile settings in the upperright corner of the interface by clicking on the Email icon. If you need to select the node address or create a new blockchain address for linking the client's account to it, please, click the *Address* button.

Client supports all types of modern browsers. If the client web interface does not work properly, or if you see any errors during loading pages, please, update your browser to the latest version.

Network stats

On the Common information tab you can see the following data:

- Network load in percent.
- Average block size in bytes.
- Number of blocks.
- Transaction senders number.
- Number of available nodes.
- Information about the latest running smart contracts and their execution time.

Explorer

This section contains information about blockchain transactions. For information, use the filter and the search string to specify the transaction fields to search for.

Available transaction filters:

- All transactions displays of all transactions.
- Data transactions operations with data transactions (*Data Transaction*).
- Tokens a selection of transactions with tokens. When this value is selected, an additional option of contextual filtering by types of token operations (for example, transfer, lease or issue of tokens) appears.
- Permissions a selection of the permissions transactions. Context filters are available by permission type (for example, mining, publishing contracts, or managing access).
- Groups a selection of privacy data access groups transactions. When this value is selected, an additional option of contextual filtering by operation types (for example, a creation or an update of the access group) appears.
- Contracts a selection of the contracts transactions. When this value is selected, an additional option of contextual filtering by Docker contracts appears.
- Unconfirmed transactions.
- Users information about users. Context filters are available by permission type (for example, mining, publishing contracts, or managing access).

Tokens

This section shows the balance of authorized account. Allows transferring tokens to other network participants, transfer tokens for lease and manage tokens. Token management requires the "Token Management" permission.

Contracts

The section displays information on existing contracts in the network and allows you to run the selected contracts. You can use the search string with transaction parameters for the filtration. Contract publishing requires the "contractdeveloper" role.

Data tranfer

The section allows to create data transactions and view information about existing data transactions. Also it is available to create privacy groups and exchange data in them.

Network settings

In this section, you can view information about nodes in the network, as well as calculate the leasing payouts. To do this, you need to specify the following data:

- Leasing Pool Address.
- Beginning and end of the payout period (blockchain height)
- Payout percentage.

The algorithm for calculating the leasing payout is as follows:

1. At the beginning of the period the generating balance is requested from the node whose address was specified as the leasing pool field.
2. The lease sum is calculated taking into account the miner's profit (the miner should get 40% for his block and 60% for the previous block).
3. This sum is divided for each participant in proportion to the amount of leasing funds and the generating balance of the node at the specified height.
4. The calculated leasing sum is multiplied by the profit percentage.
5. The node's generating balance is recalculated for the new height, taking into account new and canceled leases.

Note: Leasing funds must remain in the lease for at least 1000 blocks without movement before they begin to make a profit.

Write to us

You can send free comments and feedback to our technical support. All requests will be considered.

Address

You can go to the page by clicking the *Address* button in the upperright corner of the interface. The section contains basic information about the user's account (public and private keys, secret phrase). Also you can add permissions to another users. This option requires the "permissioner" role. If the blockchain address is not linked to the client's account, you can create it on this page or specify the node address from the key storage.

Account settings

You can go to the page by clicking the "Email" icon in the upperright corner of the interface. This section shows the current version of the client and allows you to change the language of the interface.

CHAPTER

TWELVE

BLOCKS, TRANSACTIONS, MESSAGES

12.1 Blocks

This module contains the structure of block storage in the Waves Enterprise blockchain.

Field order number	Field	Type	Field size in bytes
1	Version (0x02 for Genesis block, 0x03 for common block)	Byte	1
2	Timestamp	Long	8
3	Parent block signature	Bytes	64
4	Consensus block length (always 40 bytes)	Int	4
5	Base target	Long	8
6	Generation signature*	Bytes	32
7	Transactions block length (N)	Int	4
8	Transaction #1 bytes	Bytes	M1
...	...	...	...
8 + (K 1)	Transaction #K bytes	Bytes	MK
9 + (K 1)	Generator's public key	Bytes	32
10 + (K 1)	Block's signature	Bytes	64

Generation signature is calculated based on the hash (Blake2b256) of the following fields:

Field order number	Field	Type	Field size in bytes
1	Previous block's generation signature	Bytes	32
2	Generator's public key	Bytes	32

The block signature is calculated based on the following data:

Field order number	Field	Type	Field size in bytes
1	Version (0x02 for Genesis block,, 0x03 for common block)	Byte	1
2	Timestamp	Long	8
3	Parent block signature	Bytes	64
4	Consensus block length (always 40 bytes)	Int	4
5	Base target	Long	8
6	Generation signature*	Bytes	32
7	Transactions block length (N)	Int	4
8	Transaction #1 bytes	Bytes	M1
...	...	...	...
8 + (K 1)	Transaction #K bytes	Bytes	MK
9 + (K 1)	Generator's public key	Bytes	32

12.2 Transactions

In this section you can see the structure of transaction storage in the blockchain platform of Waves Enterprise. For some types of transactions, versioning is introduced.

12.2.1 Data format in transactions

All transactions use the `timestamp` field containing a time stamp in the **Unix Timestamp** format in milliseconds.

The *3*, *13*, *14* and *112* transactions use the `description` text field, while *4* and *6* transactions the `attachment` text field. Messages to be sent in this transaction fields should be converted into the **base58** format prior to sending.

The keyvalue fields of the `data` section in the *12* transaction, as well as those of the `params` section in the *104* transaction support 4 data types: *string*, *integer*, *boolean*, *binary*.

The proto files defining the reply format of the node are available on our GitHub page.

12.2.2 Storage format of the transactions

The values of json requests for signing and sending transactions to the blockchain are samples. Before sending a request to sign a transaction, check whether the request parameters match the current data. For example, if you are sending a transaction to Mainnet, you need to make sure that you have specified the correct transaction fee. Otherwise, the transaction will not pass validation, and the node will return the 105 `InvalidFee` error.

For more information about transaction comissions, see *Commissions on the network "Waves Enterprise Mainnet"*

Table 1: Transaction types

№	Transaction type	Description
1	*Genesis transaction*	Initial binding of the balance to the addresses of nodes created at the start of the blockchain
3	*Issue Transaction*	Tokens issue
4	*Transfer Transaction*	Tokens transfer
5	*Reissue Transaction*	Tokens reissue
6	*Burn Transaction*	Tokens burn
8	*Lease Transaction*	Tokens lease
9	*Lease Cancel Transaction*	Cancel of the tokens lease
10	*Create Alias Transaction*	Alias creation
11	*MassTransfer Transaction*	Mass tokens transfer. Minimum commission is specified
12	*Data Transaction*	Transaction with the data in the keyvalue pairs format. Minimum commission is specified
13	*SetScript Transaction*	Transaction which is binding a script with a RIDE contract to an account
14	*Sponsorship Transaction*	Transaction which is signing a sponsorship asset
15	*SetAssetScript*	Transaction which is binding a script with a RIDE contract to an asset
101	*Genesis Permission Transaction*	Assignment of the first network administrator for further distribution of rights
102	*Permission Transaction*	Issuance/withdrawal of rights from the account
103	*CreateContract Transaction*	Dockercontract creation
104	*CallContract Transaction*	Dockercontract call
105	*ExecutedContract Transaction*	Dockercontract execution
106	*DisableContract Transaction*	Dockercontract disable
107	*UpdateContract Transaction*	Dockercontract update
110	*GenesisRegisterNode Transaction*	Node registration in the genesis block with the blockchain start
111	*RegisterNode Transaction*	A new node registration
112	*CreatePolicy Transaction*	Access group creation
113	*UpdatePolicy Transaction*	Update the access group
114	*PolicyDataHash Transaction*	A data hash sending to the net
120	*AtomicTransaction Transaction*	Packaging multiple transactions into one for atomic execution

Important: The 101 transaction is also designed for enabling of the *sender* role. This role can be granted to participants with the use of the transactions 101 and 102. However, if at least one member has not been assigned as `sender' using the 101 transaction, it will not be possible to assign that role using the 102 transaction.

1. Genesis transaction

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
fee	+		Long
timestamp	+	+	Long
signature	+		ByteStr
recipient	+	+	ByteStr
amount	+	+	Long
height	+		

3. Issue Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
assetId		+		ByteStr
name	+	+	+	Array[Byte]
quantity	+	+	+	Long
reissuable	+	+	+	Boolean
decimals	+	+	+	Byte
description	+	+	+	Array[Byte] (*base58*)
chainId		+	+	Byte
script	+ (opt)	+	+	Bytes
password	+ (opt)			String
height		+		

JSON to sign

```
{
   "type": 3,
   "version":2,
   "name": "Test Asset 1",
   "quantity": 100000000000,
   "description": "Some description",
   "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
   "password": "",
   "decimals": 8,
   "reissuable": true,
   "fee": 100000000
}
```

Broadcasted JSON

```
{
    "type": 3,
    "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 100000000,
    "timestamp": 1549378509516,
    "proofs": [
↪"NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG" ],
    "version": 2,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "name": "Token Name",
    "quantity": 10000,
    "reissuable": true,
    "decimals": 2,
    "description": "SmarToken",
    "chainId": 84,
    "script": "base64:AQa3b8tH",
    "height": 60719
},
```


4. Transfer Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
recipient	+	+	+	ByteStr
assetId	+ (opt)	+	+	ByteStr
fee assetId	+ (opt)	+	+	Bytes
amount	+	+	+	Long
attachment	+ (opt)	+	+	Bytes (*base58*)
password	+ (opt)			String
height		+		
atomicBadge	+	+	+	

JSON to sign


```
{
  "type": 4,
  "version": 2,
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 40000000000,
  "fee": 100000
}
```

Broadcasted JSON


```
{
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "amount": 200000000,
    "fee": 100000,
    "type": 4,
    "version": 2,
    "attachment": "3uaRTtZ3taQtRSmquqeC1DniK3Dv",
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "feeAssetId": null,
    "proofs": [
    "2hRxJ2876CdJ498UCpErNfDSYdt2mTK4XUnmZNgZiq63RupJs5WTrAqR46c4rLQdq4toBZk2tSYCeAQWEQyi72U6"
    ],
    "assetId": null,
    "recipient": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
    "id": "757aQzJiQZRfVRuJNnP3L1d369H2oTjUEazwtYxGngCd",
    "timestamp": 1558952680800
}
```


5. Reissue Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
assetId	+	+	+	ByteStr
quantity	+	+	+	Long
reissuable	+	+	+	Boolean
password	+ (opt)			String
height				

JSON to sign


```
{
    "type": 5,
    "version":2,
    "quantity": 10000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
    "reissuable": true,
    "fee": 100000001
}
```


Broadcasted JSON

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "quantity": 10000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "chainId": 84,
    "proofs": [
↪"3gmgGM6rYpxuuR5QvJkugPsERG7yWYF7JN6QzpUGJwT8Lw6SUHkzzk8R22A7cGQz7TQQ5NifKxvAQzwPyDQbwmBg" ],
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
    "fee": 100000001,
    "id": "GsNvk15Vu4kqtRmMSpYW21WzgJpZrLBwjCREHWuwnvh5",
    "type": 5,
    "version": 2,
    "reissuable": true,
    "timestamp": 1551447859299,
    "height": 1190
}
```


6. Burn Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
assetId	+	+	+	ByteStr
quantity	+		+	Long
amount		+		Long
password	+ (opt)			String
height				

JSON to sign


```
{
    "type": 6,
    "version": 2,
    "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
    "password": "",
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
    "quantity": 1000,
    "fee": 100000,
    "attachment": "string"
}
```


Broadcasted JSON


```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "amount": 1000,
```


(continues on next page)

(continued from previous page)

```
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "chainId": 84,
    "proofs": [
↪"kzTwsNXjJkzk6dpFFZZXyeimYo6iLTVbCnCXBD4xBtyrNjysPqZfGKk9NdJUTP3xeAPhtEgU9hsdwzRVo1hKMgS" ],
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
    "fee": 100000,
    "id": "3yd2HZq7sgun7GakisLH88UeKcpYMUEL4sy57aprAN5E",
    "type": 6,
    "version": 2,
    "timestamp": 1551448489758,
    "height": 1190
}
```

8. Lease Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
amount	+	+	+	Long
recipient	+	+	+	ByteStr
status		+		
password	+ (opt)			String
height		+		

JSON to sign

```
{
    "type": 8,
    "version": 2,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
    "amount": 1000,
    "fee": 100000
}
```

Broadcasted JSON

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "amount": 1000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "proofs": [
↪"5jvmWKmU89HnxXFXNAd9X41zmiB5fSGoXMirsaJ9tNeyiCAJmjm7MR48g789VucckQw2UExaVXfhsdEBuUrchvrq" ],
    "fee": 100000,
    "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
```

(continues on next page)

(continued from previous page)


```
    "id": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "type": 8,
    "version": 2,
    "timestamp": 1551449299545,
    "height": 1190
}
```


9. Lease Cancel Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
leaseId	+ (txId)	+	+	Byte
leaseId		+		
password	+ (opt)			String
height		+		

JSON to sign


```
{
    "type": 9,
    "version": 2,
    "fee": 100000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "txId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp"
}
```


Broadcasted JSON


```
{
   "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "leaseId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "chainId": 84,
    "proofs": [
↪"2Gns72hraH5yay3eiWeyHQEA1wTqiiAztaLjHinEYX91FEv62HFW38Hq89GnsEJFHUvo9KHYtBBrb8hgTA9wN7DM" ],
    "fee": 100000,
    "id": "9vhxB2ZDQcqiumhQbCPnAoPBLuir727qgJhFeBNmPwmu",
    "type": 9,
    "version": 2,
    "timestamp": 1551449835205,
    "height": 1190
}
```

10. Create Alias Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
alias	+	+	+	Bytes
password	+ (opt)			String
height		+		

JSON to sign

```
{
    "type": 10,
    "version": 2,
    "fee": 100000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "alias": "hodler"
}
```

Broadcasted JSON

```
{
    "type": 10,
    "id": "DJTaiMpb7eLuPW5GcE4ndeE8jWsWPjx8gPYmbZPJjpag",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 0,
    "timestamp": 1549290335781,
    "signature":
↪"2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbBDzRZYEY",
    "proofs": [
↪"2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbBDzRZYEY" ],
    "version": 1,
    "alias": "testperson4",
    "height": 59245
}
```

11. MassTransfer Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
assetId	+ (opt)	+	+	ByteStr
attachment	+ (opt)	+	+	(*base58*)
number of transfers	+	+	+	List[Transfer]
transferCount		+	+	
totalAmount		+		
password	+ (opt)			String
height		+		

JSON to sign

```
{
    "type": 11,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "fee": 2000000,
    "version": 1,
    "transfers":
    [
        { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000 },
        { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc", "amount": 100000 }
    ]
}
```

Broadcasted JSON

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "fee": 2000000,
    "type": 11,
    "transferCount": 2,
    "version": 1,
    "totalAmount": 200000,
    "attachment": "",
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "proofs": [
↪"2gWpMWdgZCjbygCX5US3aAFftKtGPRSK3aWGJ6RDnWJf9hend5sBFAgY6u3Mp4jN8cqwaJ5o8qrKNedGN5CPN1GZ" ],
    "assetId": null,
    "transfers":
    [
        {
            "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB",
            "amount": 100000
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc",
            "amount": 100000
        }
    ],
    "id": "D9jUSHHcJqVAvkFMiRfDBhQbUzoSfQqd9cjaunMmtjdu",
    "timestamp": 1551450279637,
    "height": 1190
}
```

12. Data Transaction

Warning: The transaction has limits:

1. `"key":"value"` pairs count no more than 100,

```
"data": [
      {
       "key": "objectId",
       "type": "string",
       "value": "obj:123:1234"
      }, {...}
     ]
```

2. The byte composition of the signed transaction should not exceed more than 150 KB.

Hint: You do not need to specify the `senderPublicKey` parameter if you are signing a transaction where the author and the sender are the same.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size (Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
sender's public key	+ (opt)	+	+	PublicKeyAccount	3264
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+		Byte	1
authorPublicKey		+	+	PublicKeyAccount	3264
author	+	+			3264
data	+	+	+		3264
password	+ (opt)			String	32767
height		+			8

JSON to sign

```
{
    "type": 12,
    "version": 1,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "data": [
        {
        "key": "objectId",
        "type": "string",
        "value": "obj:123:1234"
        }
    ],
    "fee": 100000
}
```


Broadcasted JSON


```
{
"senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
"authorPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
"data":
[
    {
        "type": "string",
        "value": "obj:123:1234",
        "key": "objectId"
    }
],
"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"proofs": [
↪"2T7WQm5XW8cFHfiFkdDEic9oNiT7aFiH3TyKkARERopr1VJvzRKqHAVnQ3eiYZ3uYN8uQnPopQEH4XV8z5SgSwsf" ],
"author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"fee": 100000,
"id": "7dMMCQNTusahZ7DWtNGjCwAhRYpjaH1hsepRMbpn2BkD",
"type": 12,
"version": 1,
"timestamp": 1551680510183
}
```

13. SetScript Transaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
chainId		+	+	Byte
version	+	+	+	Byte
script	+ (opt)	+	+	Bytes
name	+	+	+	Array[Byte]
description	+ (opt)	+	+	Array[Byte] (*base58*)
password	+ (opt)			String
height		+		

JSON to sign

```
{
    "type": 13,
    "version": 1,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "fee": 1000000,
    "name": "faucet",
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ=="
}
```

Broadcasted JSON

```
{
    "type": 13,
    "id": "HPDypnQJHJskN8kwszF8rck3E5tQiuiM1fEN42w6PLmt",
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "fee": 1000000,
    "timestamp": 1545986757233,
    "proofs": [
↪"2QiGYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzojmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFgG" ],
    "chainId": 84,
    "version": 1,
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "name": "faucet",
    "description": "",
    "height": 3805
}
```

14. SponsorshipTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
assetId	+ (opt)	+	+	ByteStr
fee	+	+	+	Long
isEnabled	+	+	+	Boolean
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
chainId		+	+	Byte
version	+	+	+	Byte
script	+ (opt)	+	+	Bytes
name	+	+	+	Array[Byte]
description	+ (opt)	+	+	Array[Byte] (*base58*)
password	+ (opt)			String
height		+		

JSON to sign

```
{
    "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
    "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
    "fee": 100000000,
    "isEnabled": false,
    "type": 14,
    "password": "1234",
    "version": 1
}
```

Broadcasted JSON

```
{
    "type": 14,
    "id": "Ht6kpnQJHJskN8kwszF8rck3E5tQiuiM1fEN42wGfdk7",
    "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
    "senderPublicKey": "Gt55fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUophy89",
    "fee": 100000000,
    "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
    "timestamp": 1545986757233,
    "proofs": [
↪"5TfgYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzojmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFh7" ],
    "chainId": 84,
    "version": 1,
    "isEnabled": false,
    "height": 3865
}
```

15. SetAssetScriptTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
assetId	+	+	+	ByteStr
script	+ (opt)	+	+	Bytes
password	+ (opt)			String
height		+		

JSON to sign

```
{
    "type": 15,
    "version": 1,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "fee": 100000000,
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg"
}
```

Broadcasted JSON

```
{
    "type": 15,
    "id": "CQpEM9AEDvgxKfgWLH2HxE82iAzpXrtqsDDcgZGPAF9J",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 100000000,
    "timestamp": 1549448710502,
    "proofs": [
↪"64eodpuXQjaKQQ4GJBaBrqiBtmkjSxseKC97gn6EwB5kZtMr18mAUHPRkZaHJeJxaDyLzGEZKqhYoUknWfNhXnkf" ],
    "version": 1,
    "chainId": 84,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "height": 61895
}
```

101. GenesisPermitTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte	
id	+		Byte	
fee	+		Long	
timestamp	+	+	Long	
signature	+		ByteStr	
target	+	+	ByteStr	
role	+	+	String	
height				

102. PermissionTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee		+		Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+		+	Byte
target	+	+	+	ByteStr
PermissionOp			+	PermissionOp
opType	+	+		String
role	+	+		String
dueTimestamp	+ (opt)	+		Option[Long]
password	+ (opt)			String
height		+		
atomicBadge	+	+	+	

JSON to sign

```
{
   "type": 102,
   "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
   "password": "",
   "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
   "fee": 0,
   "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
   "opType": "add",
   "role": "contract_developer",
   "dueTimestamp": null,
   "version": 1,
 }
```

Broadcasted JSON

```
{
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
```

(continues on next page)

(continued from previous page)


```
  "role": "contract_developer",
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "proofs": [
    "5ABJCRTKGo6jmDZCRWcLQc257CCeczmcjmtfJmbBE7TP3KsVkwvisH9kEkfYPckVCzEMKZTCd3LKAPcN8o4Git3j"
  ],
  "fee": 0,
  "opType": "add",
  "id": "8zVUH7nsDCcpwyfxiq8DCTgqL7Q23FW1KWepB9EZcFG6",
  "type": 102,
  "dueTimestamp": null,
  "timestamp": 1559048837487,
  "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL"
}
```


103. CreateContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

The `feeAssetId` field is optional and used only for *gRPC contracts* (the field value `version = 2`).

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
sender's public key		+	+	PublicKeyAccount	3264
password	+ (opt)			String	32767
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+	+	Byte	1
fee assetId	+ (opt)			Byte	1
image	+	+	+	Array[Bytes]	32767
imageHash	+	+	+	Array[Bytes]	32767
contractName	+	+	+	Array[Bytes]	32767
params	+	+	+	List[DataEntry[_]]	32767
height		+			8
atomicBadge	+	+	+		32767

JSON to sign


```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
  "password": "",
```


(continues on next page)

(continued from previous page)

```
  "params": [],
  "type": 103,
  "version": 1,
 }
```

Broadcasted JSON

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

104. CallContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

The `contractVersion` field specifies the contract version, the `1` value is for the new contract, and the `2` value is for the updated contract. This field is only available for the second version of the transaction `"version": 2,`. The contract is updated by using the *107* transaction. When you create a contract, the *104* transaction is automatically created, this transaction is calling the contract to validate it. If the contract fails or runs with error, transactions 103 and 104 will be discarded and will not fall into the block.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
sender's public key		+	+	PublicKeyAccount	3264
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+	+	Byte	1
contractVersion	+	+	+	Byte	1
contractId	+	+	+	ByteStr	32767
params	+	+	+	List[DataEntry[_]]	32767
height		+			8
password	+ (opt)			String	32767
atomicBadge	+	+	+		32767

JSON to sign

```
{
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "fee": 10,
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "password": "",
    "type": 104,
    "params":
    [
        {
           "type": "integer",
           "key": "a",
           "value": 1
        },
        {
           "type": "integer",
           "key": "b",
           "value": 100

        }
    ],
    "version": 2,
    "contractVersion": 1
}
```

Broadcasted JSON

```
{
    "type": 104,
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 10,
    "timestamp": 1549365736923,
    "proofs": [
↪"2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPERGCiQrVr4vH"]
```

(continues on next page)

(continued from previous page)

```
    "version": 2,
    "contractVersion": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params":
    [
        {
        "key": "a",
        "type": "integer",
        "value": 1
        },
        {
        "key": "b",
        "type": "integer",
        "value": 100
        }
    ]
}
```

105. ExecutedContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
sender	+		PublicKeyAccount
sender's public key	+	+	PublicKeyAccount
fee	+		Long
timestamp	+	+	Long
proofs	+	+	List[ByteStr]
version	+	+	Byte
tx	+	+	ExecutableTransaction
results	+	+	List[DataEntry[_]]
height	+		
password	+ (opt)		String
atomicBadge	+	+	

Broadcasted JSON

```
{
    "type": 105,
    "id": "38GmSVC5s8Sjeybzfe9RQ6p1Mb6ajb8LYJDcep8G8Umj",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "password": "",
    "fee": 500000,
    "timestamp": 1550591780234,
    "proofs": [
↪"5whBipAWQgFvm3myNZe6GDd9Ky8199C9qNxLBHqDNmVAUJW9gLf7t9LBQDi68CKT57dzmnPJpJkrwKh2HBSwUer6" ],
    "version": 1,
```

(continues on next page)

(continued from previous page)


```
    "tx":
         {
            "type": 103,
            "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
            "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
            "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
            "fee": 500000,
            "timestamp": 1550591678479,
            "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
            "version": 1,
            "image": "stateful-increment-contract:latest",
            "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
            "contractName": "stateful-increment-contract",
            "params": [],
            "height": 1619
          },
    "results": [],
    "height": 1619
}
```


106. DisableContractTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
contractId	+	+	+	ByteStr
height		+		
password	+ (opt)			String
atomicBadge	+	+	+	

JSON to sign


```
{
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "password": "",
    "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
    "fee": 500000,
    "type": 106,
    "version": 1,
}
```


Broadcasted JSON


```
{
"type": 106,
```


(continues on next page)

(continued from previous page)


```
"id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
"sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
"senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
"fee": 500000,
"proofs": [
↪"5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQSz75LAWix1ok2Z6PC5ezPpjqzqnr15i3RQmaEc" ],
"version": 1,
"contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
"height": 1632
}
```


107. UpdateContractTransaction

Warning: The byte composition of the signed transaction should not exceed more than 150 KB.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
sender's public key		+	+	PublicKeyAccount	3264
image	+	+	+	Array[Bytes]	32767
imageHash	+	+	+	Array[Bytes]	32767
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+	+	Byte	1
contractId	+	+	+	ByteStr	32767
height		+			8
password	+ (opt)			String	32767
atomicBadge	+	+	+		32767

JSON to sign


```
{
  "image" : "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender" : "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "",
  "fee" : 100000000,
  "contractId" : "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
  "imageHash" : "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
  "type" : 107,
  "version" : 1
 }
```


Broadcasted JSON

```
{
 "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
 "tx":
 {
 "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
 "image":"registry.wvservices.com/we-sc/tdm-increment3:1028.1",
 "sender":"3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
 "proofs":[
↪"3tNsTyteeZrxEbVSv5zPT6dr247nXsVWR5v7Khx8spypgZQUdorCQZV2guTomutUTcyxhJUjNkQW4VmSgbCtgm1Z"],
 "fee":0,
 "contractId":"EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
 "id":"HdZdhXVveMT1vYzGTviCoGQU3aH6ZS3YtFpYujWeGCH6",
 "imageHash":"17d72ca20bf9393eb4f4496fa2b8aa002e851908b77af1d5db6abc9b8eae0217",
 "type":107,"version":1,"timestamp":1572355661572},
 "sender":"3HfRBedCpWi3vEzFSKEZDFXkyNWbWLWQmmG",
 "proofs":[
↪"28ADV8miUVN5EFjhqeFj6MADSXYjbxA3TsxSwFVs18jXAsHVaBczvnyoUSaYJsjRNmaWgXbpbduccRxpKGTs6tro"],
 "fee":0,"id":"7niVY8mjzeKqLBePvhTxFRfLu7BmcwVfqaqtbWAN8AA2",
 "type":105,
 "version":1,
 "results":[],
 "timestamp":1572355666866
 }
}
```


110. GenesisRegisterNodeTransaction

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
fee	+		Long
timestamp	+	+	Long
signature	+		Bytes
version		+	Byte
targetPubKey	+	+	
height	+		

111. RegisterNodeTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+		Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version			+	Byte
targetPubKey	+	+	+	PublicKeyAccount
nodeName	+	+	+	String
opType	+	+	+	
height		+		
password	+ (opt)			String

JSON to sign

```
{
"type": 111,
"opType": "add",
"sender":"3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
"password": "",
"targetPubKey": "apgJP9atQccdBPAgJPwH3NBVqYXrapgJP9atQccdBPAgJPwHapgJP9atQccdBPAgJPwHDKkh6A8",
"nodeName": "Node #1",
"fee": 500000,
}
```

112. CreatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
policyName	+	+	+	String
recipients	+	+	+	Array[Byte]
owners	+	+	+	Array[Byte]
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
height			+	Long
description	+	+	+	String (*base58*)
password	+ (opt)			String
version	+	+	+	Byte
atomicBadge	+	+	+	

JSON to sign

```
{
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "policyName": "Policy# 7777",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
        "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
        "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  ],
  "fee": 15000000,
  "description": "Buy bitcoin by 1c",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 112,
  "version": 1,
}
```


113. UpdatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
policyId	+	+	+	String
recipients	+	+	+	Array[Byte]
owners	+	+	+	Array[Byte]
fee	+	+	+	Long
timestamp		+	+	Long
proofs		+	+	List[ByteStr]
height			+	Long
opType	+	+	+	
description	+	+	+	String (*base58*)
password	+			String
version	+	+	+	Byte
atomicBadge	+	+	+	

JSON to sign


```
{
  "policyId": "7wphGbhqbmUgzuN5wzgqwqtViTiMdFezSa11fxRV58Lm",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
```


(continues on next page)

(continued from previous page)


```
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
      "3NwJfjG5RpaDfxEhkwXgwD7oX21NMFCxJHL"
   ],
   "fee": 15000000,
   "opType": "add",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 113,
  "version": 1,
}
```


114. PolicyDataHashTransaction

When the user sends confidential data to the network using *POST /privacy/sendData*, the node automatically will create the 114 transaction.

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+	+	Byte
sender	+	+	PublicKeyAccount
sender's public key	+	+	PublicKeyAccount
policyId	+	+	String
dataHash	+	+	String
fee	+	+	Long
timestamp	+	+	Long
proofs	+	+	List[ByteStr]
height		+	Long
version	+	+	Byte
atomicBadge	+	+	

120. AtomicTransaction

The transaction places other transaction into a container for their atomic execution. This transaction supports following transactions for containerization:

- *4 Assets transfer*, version 3
- *102 Adding / removing permissions*, version 2
- *103 Contract creation*, version 3
- *104 Contract calling*, version 4
- *105 Contract execution*, versions 1 and 2
- *106 Contract disabling*, version 3
- *107 Contract updating*, version 3
- *112 Privacy group creation*, version 3

- *113 Privacy group updating*, version 3
- *114 Privacy data edding*, version 3

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
sender's public key		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp		+	+	Long
proofs		+	+	List[ByteStr]
height			+	Long
transactions	+	+	+	
miner		+	+	String
password	+			String
version	+		+	Byte

JSON to sign

```
{
    "sender": sender_0,
    "transactions": [
        signed_transfer_tx,
        signed_transfer_tx2
    ],
    "type": 120,
    "version": 1,
    "password":"lskjbJJk$%^#298",
    "fee": 0,
}
```

Request sample

```
{'sender': '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP',
'transactions': [
    {'senderPublicKey':
↪'5nGi8XoiGjjyjbPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFwFXJYUmto5Joox76c3pp2Mr1LjgodUJuxryCJofQ',
↪'amount': 10, 'fee':    10000000, 'type': 4, 'version': 3, 'atomicBadge': {'trustedSender':
↪'3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP'}, 'attachment': '', 'sender':
↪'3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c', 'feeAssetId': None, 'proofs':   [
↪'XQ7iAqkarmm14AATc2Y9cR3Z9WnirsH4kH6RUL4QdT82rEwsmWBbBfWrADLE9o4cp2VR39W6b3vdrwFgg1dX7m3'],
↪'assetId': None, 'recipient':    '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'id':
↪'FZ59wAZnkFUqXjn61vvyj59fRa3cuS6nzuW3vqoRMsM5', 'timestamp': 1602857131666},  {'senderPublicKey
↪': '56rV5kcR9SBsxQ9LtNrmp6V72S4BDkZUJaA6ujZswDneDmCTmeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj',
↪'amount': 20, 'fee':     10000000, 'type': 4, 'version': 3, 'atomicBadge': {'trustedSender':
↪'3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP'}, 'attachment': '', 'sender':
↪'3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'feeAssetId': None, 'proofs':    [
↪'5KaXUFan2JD6VsJeGNyBCXEwqCjUF1nASAzxjnPZzBydXA5RJyXQGaL6N9MQ8GDNori1nXw5FsDLBqc3CPM3ezsk'],
↪'assetId': None, 'recipient':    '3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c', 'id':
↪'8GTqE1cc6zTVxYgQxgHJWJitVsDFRc6GmU5FJcnp5gu2', 'timestamp': 1602857132314}
    ],
'type': 120,
'version': 1}
```

12.3 Network messages

This section describes the structure of network messages in the Waves Enterprise blockchain platform.

12.3.1 Network message

All network messages, except Handshake, are based on the following structure:

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Payload	Bytes	N

Magic Bytes are 0x12, 0x34, 0x56, 0x78. Payload checksum is first 4 bytes of_FastHash_of Payload bytes. FastHash is hash function Blake2b256(data).

12.3.2 Handshake message

Handshake message is intended for primary data exchange between two nodes. An authorized Handshake contains the node owner's blockchain address and signature. Unsigned Handshakes are not accepted.

Authorized Handshake

Field order number	Field	Type	Field size in bytes
1	HandshakeType	byte	1
2	Application name length (N)	Byte	1
3	Application name (UTF8 encoded bytes)	Bytes	N
4	Application version major	Int	4
5	Application version minor	Int	4
6	Application version patch	Int	4
7	Consensus name lenght (P)	Byte	1
8	Consensus name lenght (UTF8 encoded bytes)	Bytes	P
9	Node name length (M)	Byte	1
10	Node name (UTF8 encoded bytes)	Bytes	M
12	Node nonce	Long	8
13	Declared address length (K) or 0 if no declared address was set	Int	4
14	Declared address bytes (if length is not 0)	Bytes	K
15	Peer port	Int	4
16	Node owner address	Bytes	26
17	Signature	Bytes	64

12.3.3 GetPeers message

GetPeers message is sent to request network addresses of network participants.

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x01)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4

12.3.4 Peers message

Peers message is a response to a GetPeers request.

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x02)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Peers count (N)	Int	4
7	Peer #1 IP address	Bytes	4
8	Peer #1 port	Int	4
...	...	...	...
6 + 2 * N 1	Peer #N IP address	Bytes	4
6 + 2 * N	Peer #N port	Int	4

12.3.5 GetSignatures message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x14)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block IDs count (N)	Int	4
7	Block #1 ID	Bytes	64
...	...	...	...
6 + N	Block #N ID	Bytes	64

12.3.6 Signatures message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x15)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block signatures count (N)	Int	4
7	Block #1 signature	Bytes	64
...	...	...	...
6 + N	Block #N signature	Bytes	64

12.3.7 GetBlock message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x16)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block ID	Bytes	64

12.3.8 Block message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x17)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block bytes (N)	Bytes	N

12.3.9 Score message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x18)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Score (N bytes)	BigInt	N

12.3.10 Transaction message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x19)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Transaction (N bytes)	Bytes	N

12.3.11 Checkpoint message

Field order number	Field	Type	Field size in bytes
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x64)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Checkpoint items count (N)	Int	4
7	Checkpoint #1 height	Long	8
8	Checkpoint #1 signature	Bytes	64
...	...	...	...
6 + 2 * N 1	Checkpoint #N height	Long	8
6 + 2 * N	Checkpoint #N signature	Bytes	64

CHAPTER

THIRTEEN

DOCKER SMART CONTRACTS

The Waves Enterprise platform provides the ability to develop and use Turingcomplete smart contracts.

13.1 Smart contracts on the Waves Enterprise platform

Turingcomplete smart contracts allow you to implement any logic embedded in the program code. To separate the launch and operation of smart contracts themselves from the Waves Enterprise platform, Dockerbased containerization is used. However, any programming language can be used to write a smart contract. Each smart contract is run in a Docker container to isolate its operation and manage the resources of the running smart contract.

When a smart contract is launched in a blockchain network, its code cannot be arbitrarily changed, replaced, or prohibited from being executed without interfering with the entire network. This property makes smart contracts an almost irreplaceable tool in the blockchain network.

Docker Registry is used for storing smart contracts with read access to Docker images for machines with nodes. Waves Enterprise provides an open repository for Docker smart contracts, where any developer can add their own smart contract. The open repository is located at the `registry.wavesenterprise.com/wavesenterprisepublic` address. To add your smart contract to the open repository, you need to write a request to our technical support. After the request is approved, the smart contract will be added to the open repository, and you can call it from the client or the node's REST API.

If you use a private blockchain network, you need to have your own Docker repository for publishing and calling smart contracts.

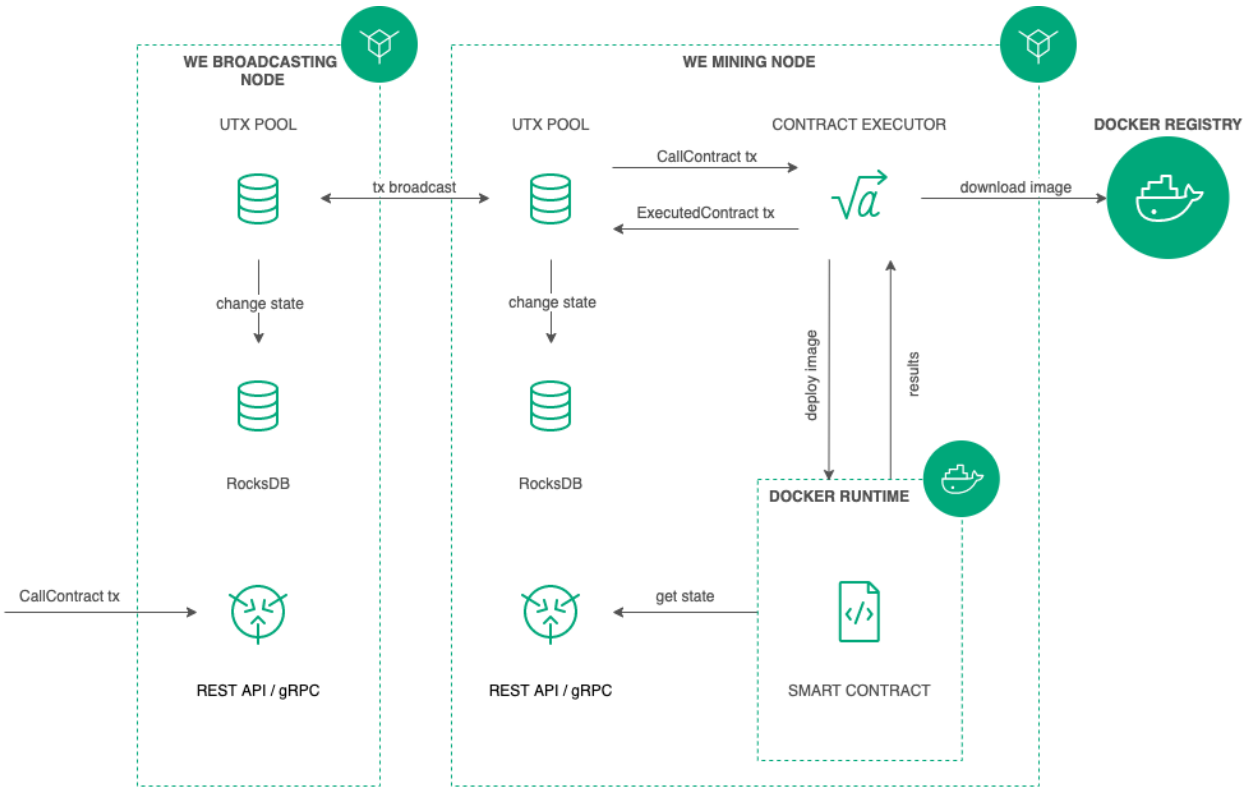
The node state can be accessed through a REST API or gRPC.

A smart contract can be created and called by any network participant, regardless of whether there is a mining node or not. You need just to register in the Mainnet network via *client interface*.

13.2 Creating a contract

Creating a smart contract starts with the preparation of a Docker image, which consists of the contract program code, the required environment, and the special scenario Dockerfile. A prepared Docker image (a build) is then assembled and sent to Docker Registry. To send the new smart contract, create a request on the technical support portal. After verifying the smart contract, technical support staff places it in an open Docker repository. The settings of the `dockerengine` section for working with the open Docker repository are already presented by default in the *node configuration file*. Also the recommended parameter values are set by default for optimal operation of smart contracts in the Mainnet blockchain network.

Dockerfile sample for REST API usage:




```
FROM python:alpine3.8
ADD contract.py /
ADD run.sh /
RUN chmod +x run.sh
CMD exec /bin/sh -c "trap : TERM INT; (while true; do sleep 1000; done) & wait"
```

Dockerfile sample for gRPC usage:

```
FROM python:3.9-rc-buster
RUN pip3 install grpcio-tools
ADD src/contract.py /
ADD src/protobuf/common_pb2.py /protobuf/
ADD src/protobuf/contract_pb2.py /protobuf/
ADD src/protobuf/contract_pb2_grpc.py /protobuf/
ADD run.sh /
RUN chmod +x run.sh
ENTRYPOINT ["/run.sh"]
```

The contract is created by publishing a special (CreateContractTransaction) transaction containing a link to the image in Docker Registry. To use the REST API or gRPC, please, specify the transaction version 103. After the transaction is received, the node downloads the image using the link specified in the “image” field, the image is checked and launched as a Docker container.

13.3 Executing a Contract

Smart contract execution is initiated by a special (CallContractTransaction) transaction containing the contract ID and call parameters. The transaction ID defines the Docker container. The container is executed unless it has been launched before. The contract launch parameters are transferred to the container. | Smart contracts change their state by updating the keyvalue pairs.

13.4 Parallel contract execution

The Waves Enterprise platform allows you to run multiple Docker contracts simultaneously. This option is only supported by *gRPC contracts*. How it works:

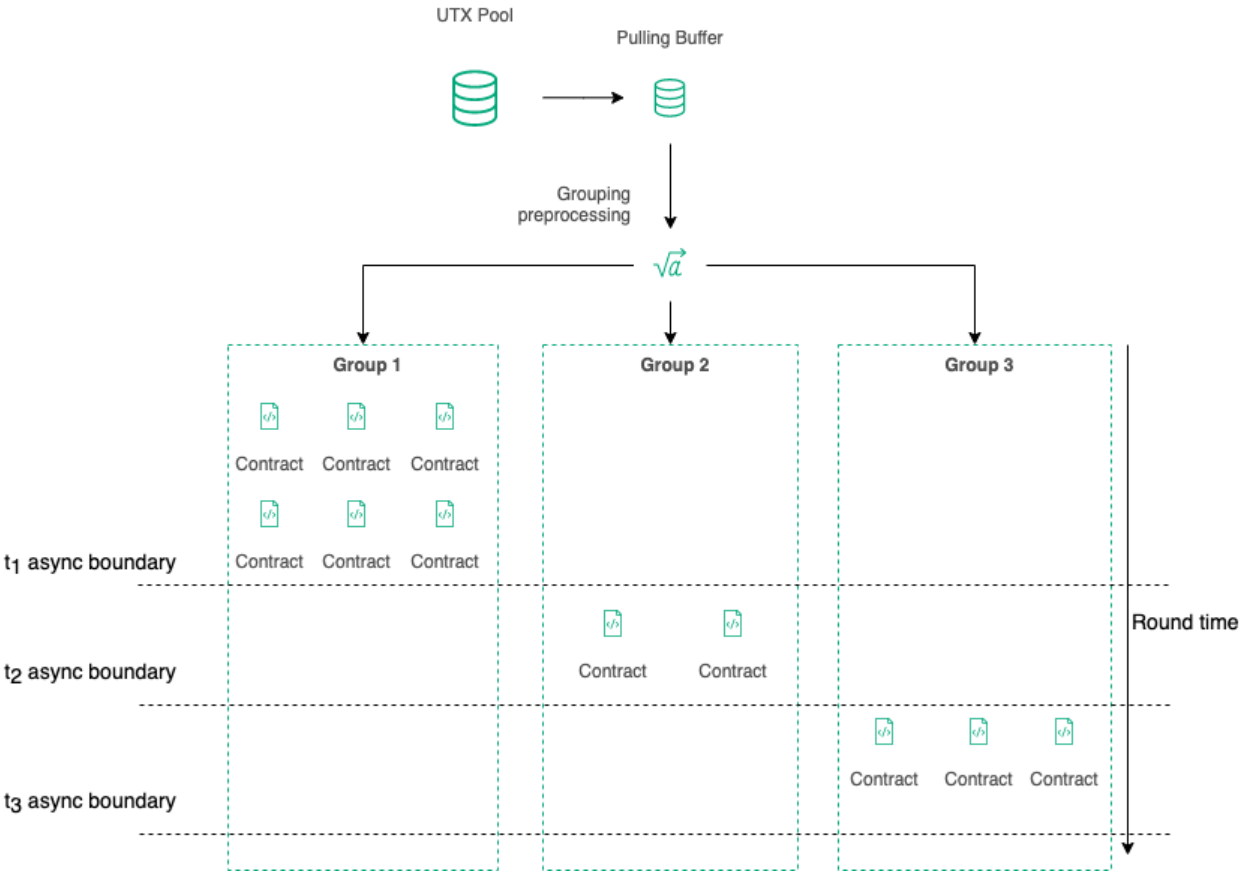
1. The developer of the smart contract specifies an `asyncfactor` parameter in the contract code (for more information, see *Creating a smart contract*). This parameter defines the allowed number of simultaneous transactions for a smart contract.
2. Right after starting the contract passes the value of the `asyncfactor` parameter to the node.
3. When contract execution starts, the buffer for contracts starts filling up. Raw transactions with contracts are taken from the UTX pool until the buffer is full.
4. Then the selected transactions are divided into groups by contract IDs. Only one group can be executed at a time and within the group, parallel contract processing is defined by the `asyncfactor` parameter.
5. When the next contract goes to execution, one cell in the buffer is released, and when a transaction is received from the UTX pool, the cell is blocked. In this way, buffer filling and contract call processing operations occur in parallel, which avoids pauses for pulling transactions up when all current transactions have already been processed.
6. Parallel execution of contracts is affected by the value of the `pullingbuffersize` parameter. This parameter is configured in the *dockerengine* section of the node configuration file and indicates the buffer size for processing transactions with contracts.

The diagram below shows an approximate principle of parallel processing of smart contracts.

The buffer allows you to have a stock of contract transactions which are ready for processing. In addition, the buffer size affects the maximum amount of time that can be used to process contracts from a single group. The larger the buffer size, the longer this time is. Therefore, the buffer size is also a limit for the number of transactions waiting to be processed. If this limit is exceeded, transaction processing is moved to the next group.

The smart contract code's logic, as well as the selected development tools (the programming language in which the contract is written), should take into account the specifics of parallel processing of the contract. For example, if a smart contract with the increment function is executed in parallel, the result is incorrect because a shared key is used during each contract call. A contract that implements the voting procedure on the Waves Enterprise platform does not use shared keys and supports parallel execution, which increases the efficiency of its processing and guarantees faster results.

Read more about creating a smart contract on the Creating a smart contract page.


UTX Pool
Pulling Buffer
Grouping preprocessing
Group 1
Group 2
Group 3
Contract
Contract
Contract
Contract
Contract
Contract
Contract
Contract
Contract
Contract
Contract
Contract
t1 async boundary
t2 async boundary
t3 async boundary
Round time

13.5 Updating Contract

Only the developer of the Docker smart contract can update this contract. The developer should keep the `contract_developer` *role* during the contract update and should be the *103* transaction creator. *107* transaction is using for the contract update. And it is necessary that the contract is active.

All the mining nodes download the contract image and run it for the checking after the *107* transaction includes into the block. Then the *105* transaction is issued within the *107* transaction inside it.

13.6 Contract Call Disabling

If necessary, the contract developer can disable calling the contract. To do this, a special (DisableContract-Transaction) transaction is published specifying the Contract ID. The contract becomes unavailable after its disconnection, but you can get information about the contract from the the blockchain later.

13.7 Description of Transactions

The following transactions are implemented to ensure the interaction between the blockchain and the Docker Contract:

Code	Transaction type	Purpose
103	*CreateContractTransaction*	Initiates the Contract. Transaction is signed by a user with the role *"contract_developer"*
104	*CallContractTransaction*	Calls the Contract. Transaction is signed by the initiator of contract execution
105	*ExecutedContractTransaction*	Records the contract execution result in the contract state. |**br**| Transaction is signed by the block generating node
106	*DisableContractTransaction*	Disables calling a contract. |**br**| Transaction is signed by a user with the role *"contract_developer"*
107	*UpdateContractTransaction*	Updates a contract. |**br**| Transaction is signed by a user with the role *"contract_developer"* |**br**| Only the contract developer and *103* transaction issuer can update the contract

13.8 Node configuration

Downloading and execution of Docker Contracts initiated by transactions with codes `103107` are performed on nodes with enabled option `dockerengine.enable = yes` (for details see module *"Node configuration"* > *"Docker configuration"*).

13.9 REST API

The REST API methods description for the Docker contract usage is represented on the *API methods available to smart contract* page.

13.10 gRPC

The gRPC methods description for the Docker contract usage is represented on the *gRPC services available to smart contract* page.

13.11 Implementation examples

- Creating a simple contract

CHAPTER

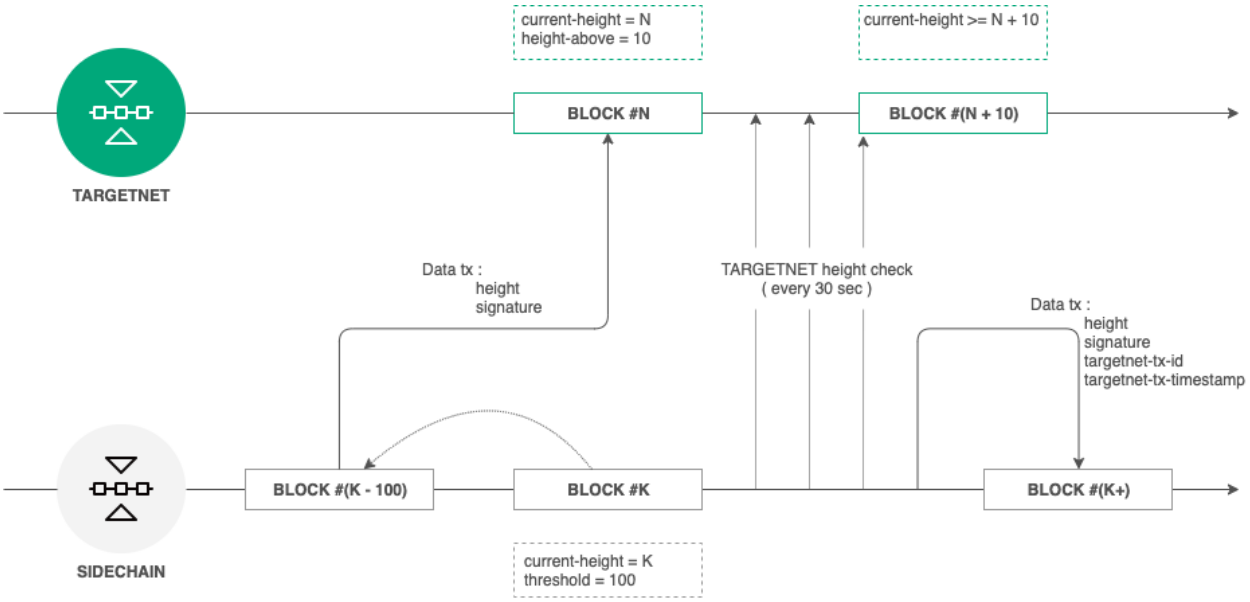
FOURTEEN

ANCHORING

In a private blockchain, transactions are processed by a certain number of participants known in advance. Thus, there is a threat of information spoofing, because the number of participants is quite small compared to a public blockchain where anyone can join the network. When using PoS consensus algorithm in a private blockchain, the threat of overwriting that blockchain becomes real.

The anchoring mechanism was developed to increase participant confidence in the date placed in a private blockchain. Anchoring checks the data in a blockchain for invariability, which is achieved by publishing data from a private blockchain to a public one, where data spoofing is unlikely due to the larger number of participants and blocks. Published data represents a signature and a height of blocks in a private network. This connectivity between two or more networks increases their resistance, because any attempt to forge or alter data using a longrange attack would require attacking all connected networks.

14.1 How does anchoring work in the Waves Enterprise blockchain



Anchoring process is shown below:

1. *Anchoring configurations* are set in the configuration file of the private blockchain node. Users should use recommended values for configurations to avoid anchoring malfunctioning.
2. Each `heightrange` is an anchoring transaction that contains block data at `currentheight threshold` and is broadcasted to the Targetnet by the anchoring node. The *Data Transaction* with a `keyvalue`

list is used as *an anchoring transaction*. The node then requests height of the broadcasted transaction.

3. The node then checks the Targetnet height each 30 seconds until its height reaches **the height of the created transaction** + `heightabove`.
4. When the required Targetnet height is reached and the presence of previously created data transactions are confirmed, another anchoring data transaction is created in the private blockchain.

14.2 Transaction structure for anchoring

Targetnet transaction consists of the following fields:

- `height` the height of the chosen block from the private blockchain.
- `signature` the signature of the chosen block from the private blockchain.

The private blockchain transaction consists of the following fields:

- `height` the height of the chosen block from the private blockchain.
- `signature` the signature of the chosen block from the private blockchain.
- `targetnettxid` the Targetnet anchoring transaction ID.
- `targetnettxtimestamp` дата и время создания транзакции для анкоринга в Targetnet.

14.3 Errors during the anchoring

Errors can occur at any step during anchoring. In case of any error in the private blockchain the *Data Transaction* containing the error code and the description is always published. The error transaction includes the following data:

- `height` the height of the chosen block from the private blockchain.
- `signature` the signature of the chosen block from the private blockchain.
- `errorcode` the error code.
- `errormessage` the error message.

Table 1: Error types

Code	Message	Possible cause
0	`Unknown error`	An unknown error occurred during the send of the transaction to the Targetnet
1	`Fail to create data transaction for Targetnet`	Creating of the transaction to be sent to the Targetnet failed
2	`Fail send transaction to Targetnet`	The transaction publication to the Targetnet failed (it could be a JSON request error)
3	`Invalid http status of response from Targetnet transaction broadcast`	The Targetnet has returned an HTTP code other than 200 after the transaction publication
4	`Fail to parse http body of response from Targetnet transaction broadcast`	The Targetnet has returned an unknown JSON after the transaction publication
5	`Targetnet return transaction with id='$TargetnetTxId' but it differ from transaction that we sent id='$sentTxId`	The Targetnet has returned mismatched ID after the transaction publication
6	`Targetnet didn't respond on transaction info request`	The Targetnet has not responded to the request about the transaction info
7	`Fail to get current height in Targetnet`	Failed to get current Targetnet height
8	`Anchoring transaction in Targetnet disappeared after height rise enough`	The anchoring transaction has disappeared from the Targetnet after its height evened `heightabove` value
9	`Fail to create sidechain anchoring transaction`	Fail to public the anchoring transaction in the private blockchain
10	`Anchored transaction in sidechain was changed during Targetnet height arise await, looks like a rollback has happened`	Anchored transaction in sidechain was changed during Targetnet height arise await, looks like a rollback has happened

CHAPTER

FIFTEEN

INTEGRATION SERVICES

15.1 Authorization service

The authorization service is an external service that provides authorization for all components of the blockchain network. This service is built using the OAuth 2.0 authorization protocol. OAuth 2.0 is an open framework for realization of the authorization mechanism, allowing third parties limited access to protected resources without transferring credentials to the third party. The data flow scheme between participants sharing information using OAuth 2.0 is presented below.

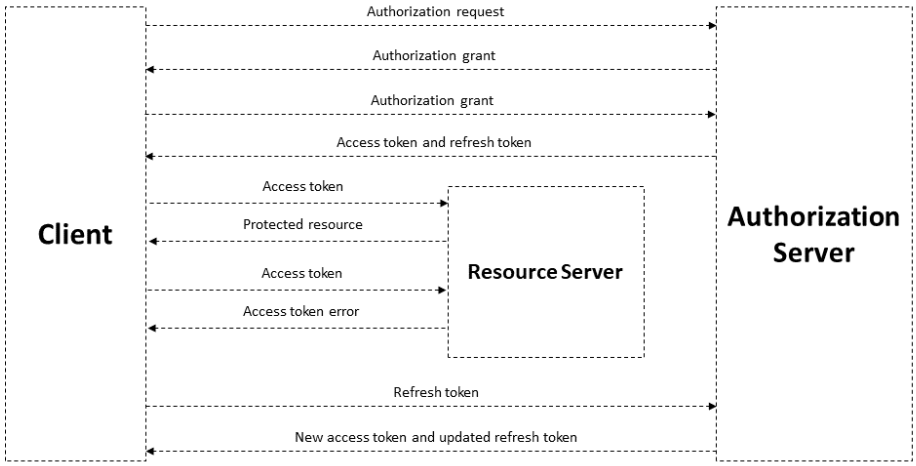


Fig. 1: Basic authorization scheme based on OAuth 2.0 protocol

A JSON Web Token is used to authorize each request from the client to the server and has a limited lifetime. The client can receive two types of tokens: access and refresh. The access token is used to authorize requests for access to protected resources and to store additional information about the user. The refresh token is used to obtain a new access token and to refresh the refresh token.

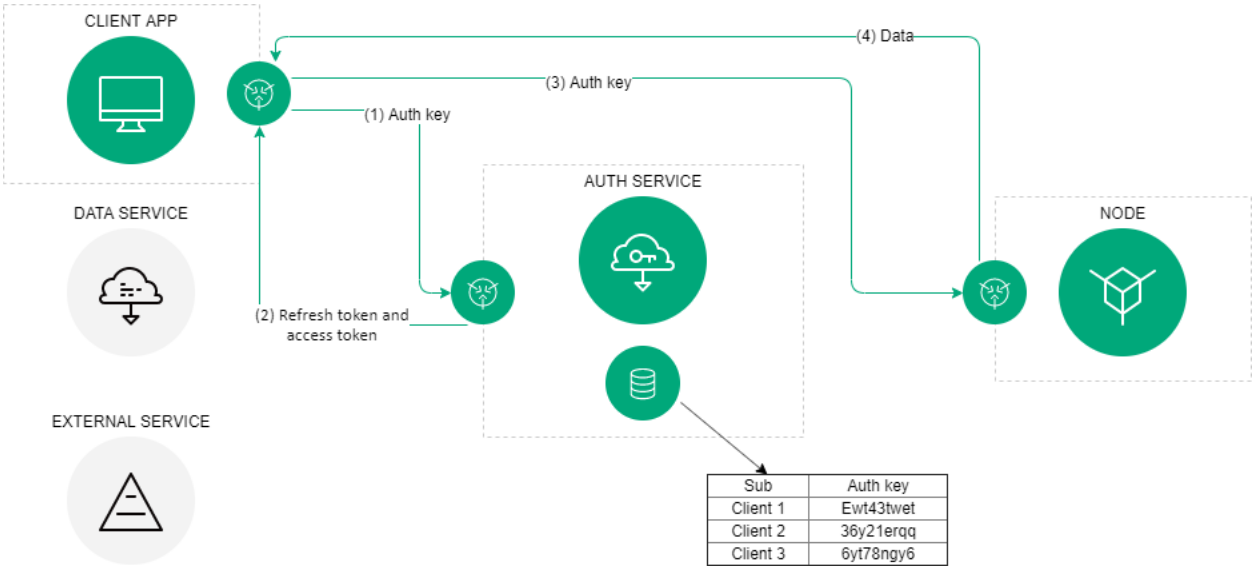


Fig. 2: The authorization scheme of the Waves Enterprise blockchain platform

In general, the authorization scheme includes the following operations:

1. The client (which could be any blockchain network component like the web client, data service, or an external application) provides its authentication data to the authorization service once.
2. If the initial authentication procedure is successful, the authorization service stores the client's authentication data in the database, generates and sends signed access, and refresh tokens to the client. Tokens include the lifetime info and basic customer data, such as an ID and a role. Client authentication data is stored in the authorization service configuration file. The client checks the lifetime of the access token each time before sending a request to a thirdparty service. In case the token is expired, the client refers to the authorization service to obtain a new access token. The refresh token is used for requests to the authorization service.
3. The client sends a request to receive data from a thirdparty service using the current access token.
4. The external application checks the lifetime of the access token and its integrity, then compares the previously obtained public key of the authorization service with the key contained in the signature of the access token. If the token is successfully verified, the service provides the requested data to the client.

15.2 Data preparation service

This service aggregates data from a blockchain into a relational database and provides an API to access that data. Service features are designed to meet the needs of the Waves Enterprise client. Specifying parameters are available for requests.

Deploy your client and node using the delivery set for service usage. Currently, access to the Data Preparation Service API is limited in the public network. The data service REST API is represented in the *Data service REST API* service.

CHAPTER

SIXTEEN

SYSTEM REQUIREMENTS

System and hardware requirements are given below.

Optional	vCPU	RAM	SSD	JVM Operation Mode
Minimum requirements	2+	2Gb	50Gb	java Xmx2048M jar
Recommended requirements	2+	4+ Gb	50+ Gb	java Xmx4096M jar

Hint: "Xmx" flag defining maximum size of memory available for JVM.

Waves Enterprise platform environment requirements

- Oracle Java SE 11 (64bit) or OpenJDK 11 and higher
- Docker CE
- Dockercompose

CHAPTER

SEVENTEEN

INSTALLING AND RUNNING THE PLATFORM

Currently we support Unixlike systems (for example, popular Linux distributives and MacOS). However Waves Enterprise platform can be run under the Windows natively in experimental mode. Also you can use Unix virtual machines and the Docker environment for the installation and running the platform under the Windows.

Installation of the platform in the base delivery version assumes that Docker Engine and Docker Compose are installed in the deployment environment.

Important: Make sure that you have a Docker Engine version that supports the dockercompose file format version 3.0 or higher. You can find out more on the official Docker page.

17.1 Variants of platform installation

Depending on the Waves Enterprise platform usage scenario, we offer several installation options:

17.1.1 Deploying the platform in Sandbox mode

In the trial mode you can interact with the blockchain through the client application, or REST/gRPC node interfaces: send transactions, receive data from the blockchain, set and call smart contracts, and transfer confidential data between nodes.

1. Создайте рабочую директорию и поместите туда файл `dockercompose.yml`, который вы можете скачать с официальной страницы Waves Enterprise на GitHub, выбрав самый свежий релиз. Также скачать этот файл вы можете при помощи wget в терминале:

```
wget https://github.com/waves-enterprise/WE-releases/releases/download/v1.5.0/docker-compose.yml
```

2. To install the platform in Sandbox mode, open the terminal and go to the directory where the file `dockercompose.yml` is located, and execute the following command:

```
docker run --rm -ti -v $(pwd):/config-manager/output wavesenterprise/config-manager:v1.5.0
```

After the platform is deployed, all created passwords and addresses are stored in the `credentials.txt` file, which is located in the working directory.

3. Wait for the results of the previous command and run the following command:

```
docker-compose up -d
```

Attention: On Linux, you may need to have root right to execute commands.

After launching the containers, the client application will be available at `http://localhost`, swagger host of the node `http://localhost/node0`.

To stop running nodes and services, execute the following command:

```
docker-compose down
```

The network will operate till the 30 000 blocks height without a license. It is not necessary to get *license* for the sandbox mode.

The Waves Enterprise team offers a fully automated deployment mode to familiarize yourself with platform capabilities. n this mode, a blockchain network of three nodes will be installed as well as additional components authorization service and *corporate client*. All key pairs used to sign transactions and blocks will be generated randomly. Sandbox mode allows you to test all available options and features of the platform. The network operates up to a height of 30,000 blocks.

17.1.2 Connecting a single node to the Mainnet network

Full instructions for the node connection to the Mainnet are provided on the *Connection of the node to the "Waves Enterprise Mainnet"* page.

1. Create a working directory and place there the `dockercompose.yml` file. This file you can download from the official Waves Enterprise GitHub page choosing the latest release.
2. Put the node configuration file named `private_network.conf`. You should use the Mainnet configuration file named `mainnet.conf` from the Waves Enterprise GitHub official page. Please, download it and name it as `private_network.conf`.
3. Run the following command and wait execution results:

```
docker run --rm -ti -v $(pwd):/config-manager/output/ wavesenterprise/config-manager:v1.4.0
```

After the platform is deployed, all created passwords and addresses are stored in the `credentials.txt` file, which is located in the working directory.

4. If you have a *license* file place it in the `working directory/configs/nodes/node0/license` directory, which is creating in the working directory during the node deploy.
5. Run the command to start the node:

```
docker-compose up -d node-0
```

After the container is launched node REST API will be available at `http://localhost:6862`.

Attention: If there are errors, make sure that no other competing containers or programs are running. To display a list of running containers and their status, type `docker ps a`. To stop the selected container, enter `docker stop [myContainer]`. To stop all containers, you can enter `docker stop $(docker ps a q)`. The command `docker rm [myContainer]` will delete the selected one, `docker rm $(docker images a q)` will delete all containers.

To stop running node execute the following command:

```
docker-compose down
```

It is enough to install one node for the Waves Enterprise Mainnet connection. Full connection procedure is represented in the *Connection of the node to the "Waves Enterprise Mainnet"* page.

For a full deployment of the blockchain network from N nodes contact our technical support for getting a consultation.

17.2 Useful information for installing and operation of the platform

This section also provides such useful topics as:

17.2.1 First steps after the Waves Enterprise platform installation

One of the first things you can do after deploying the Waves Enterprise platform is to do the following steps:

- *Attaching the node address to the client*
- *Sending transactions*
- *Platform options activation*

Attaching the node address to the client

After the blockchain platform has started, follow these steps:

1. Open a browser and input `http://localhost` in the address bar.
2. Register in the client using any valid email address and log in to it.
3. Open the `Choose address` > `Create address` page. To open the menu after the first login, you should enter the password that you entered during your account registration.
4. Choose the `Add address from node keystore` option and press "Next".
5. Fill in the fields below. You can take the values from the file `credentials.txt` for the first node in the working directory.
    - *Address name* specify the name of the node.
    - *Node URL* specify the `http://localhost/nodeAddress` value.
    - *Node authorization type* choose the node authorization type (token or apikey).
    - *Blockchain address* specify the name of the node.
    - *Key pair password* specify the node key pair password.

Now you can send transactions from the web client of the node which has tokens.

Sending transactions

Transactions can be sent from the web client or using the node's REST API. You can perform the following actions via the client:

- Operations with tokens. You need *to attach the node address* to the web client for the token operations.
- Operations with private groups to exchange confidential data.
- Operations with Docker contracts.
- Using the anchoring option.
- Sending data transactions.

All actions are performed in an intuitive and friendly web interface. Each action is accompanied by sending the corresponding transaction to the blockchain.

You can use the node's REST API to send any transaction to the blockchain. Follow these steps for sending a transaction via the node's REST API:

1. Open the node's REST API using http://localhost/node-0 address in a browser.
2. Enter the apikey into the *API key authorization* form and press `Authorize`. You can copy the apikey value from the "API key" field of the `credentials.txt` file.
3. Choose *Transactions* methods, then *POST /transactions/signAndBroadcast* method and press "Try it out".
4. Using the *transactions table* choose a transaction which you want to send to the blockchain.
5. Make a json request using your parameters and examples from the *transactions* page for the each kind of transaction. Mainly these parameters are:
    - `sender` an address of a nodesender;
    - `password` a password of the `keystore.dat` file;
    - `recipient` an address of a noderecipient;
    - various identifiers.
6. Insert the request in the corresponding `body` form of the REST API, where you can also find examples of requests for sending transactions to the blockchain.
7. Press "Execute" and watch the result in the `Response body` field. The successful response code is 200.

Activating additional options

Two options are enabled in sandbox mode by default working with Docker contracts and mining. Authorization is set by *apikeyhash*. The node configuration file already contains the default settings for the local Docker host that you can develop Docker contracts immediately. Mining settings are also set by default in accordance with the recommended values.

Additional options for the Waves Enterprise platform are enabled and configured using the appropriate sections of the node configuration file. Go to the configuration file of the node where you want to enable additional options or configure the ones used, and edit the sections of the selected options.

- *Docker configuration*
- *Anchoring settings*
- *Mining settings*
- *Authorization settings*

- *Privacy groups settings*

Node configuration files are stored in individual directories of each node, for example `../working directory/configs/nodes/node0/node.conf`. Depending on the configuration file section the recommended values are either already set in the sample files, or they can be found on the section description page. The section should be uncommented or copied from the documentation from the corresponding description page.

If you have any questions about configuring sections of the node configuration file, contact technical support.

17.2.2 Updating a Mainnet node

If you are working with the Mainnet we recommend updating the connected nodes on time. After the new release issued, all clients receive a notification email about updating of a Mainnet node. If you do not receive such an email, please send a request to our technical support.

The instructions below are intended for nodes that are deployed and run using the `dockercompose.yml` file. Please, contact our technical support if you have other node versions for update.

Perform the following actions for the node update:

1. Download the latest version of the `dockercompose.yml` file from the official Waves Enterprise GitHub page choosing the latest release.
2. Place the `dockercompose.yml` file into the working directory.
3. If your node is up than stop it by the command:

```
docker-compose down
```

4. After node was down perform the command:

```
docker-compose up -d node-0
```

Attention: When you first launch the 1.4.0 release node, the data migrator will be launched. Migration is performed automatically and takes several minutes. If the migration was successful, you will see the message "Migration finished successfully" and the node will continue to run.

17.2.3 Working in web client

Using *web interface* a user performs basic operations with the blockchain. In this section, we will look at the most popular operations through it.

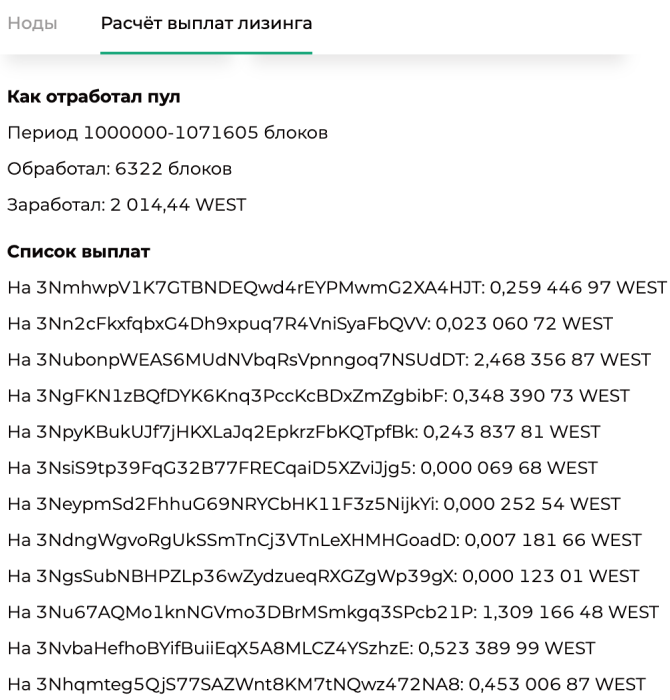
- *Calculation of lease payouts*
- *Publishing and calling a contract*
- *Sending a data transaction*
- *Working with privacy groups*

Calculation of lease payouts

You can read about the algorithm for the calculation of lease payouts on the client page. Leasing income is calculated on the page **Network settings** > **Calculation of lease payouts**. Fill in the following fields:

- Leasing Pool Address.
- Beginning of the payout period (blockchain height), but no deeper than 200,000 blocks from the current height.
- End of payout period (blockchain height). By default, the current height of the blockchain is taken.
- Payout percentage.

Нажмите кнопку **Рассчитать выплаты**. Ниже на страничке появится результат расчёта выплат.



You can also upload information about the calculation of lease payouts in *json* format for mailing to interested addresses.

Publishing and calling a contract

To work with contracts you need the `contract_developer` role. The contract samples are collected on the contracts tab and were not published in the blockchain. This data is taken directly from the Docker repository. **Published contracts** tab contains a list of all contracts created and published in the blockchain.

Using the client interface you can publish and call only the second version of *contracts*, which are working through gRPC. However, the **Contracts** page shows contracts of all versions. The contract version is indicated in its card, which is called when you click on an entry in the list of published contracts.

Follow these steps to publish the contract in the blockchain:

1. On the **Contract images** tab select the contract you want to publish to the blockchain and open its card.
2. In the contract card go to the **Publication** tab and fill in the *Contract name* field.

3. In the *New keyvalue pair* field specify as many value pairs as you want the contract to process. Select the data type for each pair (string, integer, Boolean, base64 binary data). This data should be specified in accordance with the logic of the contract code. After entering all the value pairs, click *Next*.
4. Check that all the entered data is correct.
5. For the publication of the contract then click *Next*. The contract will be published.
6. Then you can publish another contract image with new values again, or go back to the list of contracts.

Immediately after publication in the blockchain the container with the contract is assigned with an ID. This container can be called and updated later using this ID. You can create any number of contract containers from a single contract image in the blockchain repository.

Follow these steps to call an already published contract:

1. On the **Published contracts** tab select the contract you want to call and open its card.
2. Go to the **Call** tab and enter in the *New keyvalue pair* field as many value pairs as you want the contract to process. Select the data type for each pair (string, integer, Boolean, base64 binary data). This data should be specified in accordance with the logic of the contract code. After entering all the value pairs, click *Next*. The contract will be called.
3. Then you can call this contract again with different values or go back to the list of contracts.

The publication of a contract depends on its size. If the contract code is large enough, the transaction with the contract will fall into the blockchain within about 510 minutes.

Sending a data transaction

On the page **Data transfer** you can send transactions with data in keyvalue format. Follow these steps to create a data transaction on the **Record tab**:

1. Click *Create data transaction*.
2. In the *New keyvalue pair* field, specify as many value pairs as you want to fit in the transaction. In total, you can add up to 100 keyvalue pairs. Select the data type for each pair (string, integer, Boolean, base64 binary data). After entering all the value pairs, click *Next*.
3. Check that all the entered data is correct and click *Next*. The data transaction is published.

Working with privacy groups

On the **Groups** tab you can create private data exchange groups. For more information about access groups, see *Data privacy*. To work with data in private groups you need to add the address of the current node of the blockchain network. Your email address should also have the `privacy` role in the client. Contact the administrator of the authorization service to get this role.

Follow these steps to link the node address to the client's account:

1. Open the account address management form by clicking the **Address not selected** button or the name of an already linked address in the upperright corner of the interface.
2. Click **Add address** and select *Add address from the node keystore*.
3. Fill in the following fields:
    - Address name.
    - Node URL.
    - Node authorization type. The authorization type should match the type set on the node.

- Blockchain address.
- Key pair password.

4. Click **Next** to link the node address to your account.

Groups are created on the "Data transfer Groups" tab. Follow these steps to create a new private group:

1. Click the *New group* button.
2. Enter the name of the group and add the addresses of participants to the private group. If desired, you can add a description of the group. For each participant you can choose one of two roles: data access or participants management. To exchange messages within a group, each address should belong to a blockchain network node and have the *data management* role in this group. Users with a client address in the blockchain network can be members of groups, but they do not have the ability to exchange messages within the group itself.
3. If necessary, you can add a moderator to the group. The moderator has the ability to edit the group participants and does not have access to the data itself, unlike a user with the participants management role.

Note: Deleting a group from the blockchain is not possible, you can only exclude all participants from the group.

---

4. Click the *Next* button and check that the data you entered is correct. If necessary, edit the group data.
5. Click *Next* to create a private data access group.

Messages are created and stored on the "Messages" tab in the private data group card. The card opens when you click on an entry about a private data group.

CHAPTER

EIGHTEEN

MANUAL NODE CONFIGURATION

The node configuration includes the following steps:

18.1 Preparation of configuration files

These following configuration files are used for the configuration:

- `accounts.conf` – the configuration file for the accounts creation.
- `apikeyhash.conf` – the configuration file for the `apikeyhash` and `privacyapikeyhash` values creation when you choose the `apikey` string hash authorization.
- `node.conf` – the main node configuration file defining the operational principals and an option list.

18.1.1 `accounts.conf` configuration file for the accounts creation

When specifying a path, use the "forward slash" `/` as a delimiting character for directory hierarchy levels. During Linux using the value `wallet` must match the directory structure of the operating system, for example `/home/contract/we/keystore.dat`. During node setting it is prohibited to use cyrillic symbols for specifying paths to the working directory, keystore, etc.

```
// accounts.conf listing

accounts-generator {
  waves-crypto = yes
  chain-id = V
  amount = 1
  wallet = ${user.home}"/node/keystore.dat"
  wallet-password = "some string as password"
  reload-node-wallet {
    enabled = false
    url = "http://localhost:6862/utils/reload-wallet"
  }
}
```

The description of the configuration file parameters is represented below.

- `wavescrypto` – the choice of a cryptographic algorithm ("yes" use *cryptography Waves*, "no" use *GOSTcryptography*);
- `chainid` – an identifying byte of the network, the value will be necessary further on for entry in parameter `addressschemecharacter` of the node configuration file;
- `amount` – a number of generated key pairs;

- `wallet` – the path to the key storage directory on the node, the value will be required further on for entry in parameter `wallet > file` of the node configuration file. For the Waves cryptography, the path to file `keystore.dat` is specified (example, `${user.home}/nodeName/keystore.dat`), for the GOSTcryptography the path to directory (`${user.home}/nodeName/keystore/`);
- `walletpassword` – a password for access to closed node keys, the value will be necessary further for entry into the parameter `wallet > password` of the node configuration file;
- `reloadnodewallet` – an option to update the node keyStore without restarting the application, by default it is turned off (`false`). `url` parameter specifies the path to the `/utils/reloadwallet` method of the REST API node.

18.1.2 `apikeyhash.conf` configuration file

`apikeyhash.conf` configuration file is intended only for the `apikeyhash` and `privacyapikeyhash` values creation when you choose the `apikey` string authorization.

```
// api-key-hash.conf listing

apikeyhash-generator {
  waves-crypto = yes
  api-key = "some string for api-key"
}
```

Parameters description

- `wavescrypto` – the choice of a cryptographic algorithm ("yes" use *cryptography Waves*, "no" use *GOSTcryptography*);
- `apikey` – the key you need to come up with. The value of this key will need to be specified in requests to REST API node (for more details see page *REST API*).

18.1.3 `node.conf` node configuration file

Если планируется подключение к существующей сети, то для упрощения подключения запросите готовый конфигурационный файл ноды у одного из участников сетевого взаимодействия или у администратора вашей сети. При создании сети с нуля или подключении к сети "Waves Enterprise Mainnet" пример конфигурационного файла ноды можно взять на странице проекта на GitHub.

Файл `node.conf` выполнен в формате HOCON.

Об изменениях в конфигурационном файле ноды можно почитать в разделе *Изменения в конфигурационном файле ноды*.

Warning: If your node's version is 1.0 and higher you need to specify the following parameter in the `node` section of the node configuration file:

```
"features": {
"supported": [100]
}
```

This option becomes active when the total quantity of blocks from `featurecheckblocksperiod = 15000` and `blocksforfeatureactivation = 10000` parameters is achieved (25 000 of blocks). These parameters are stored in the `blockchain` section and can not be changed during Mainnet or Partnernet connection. Nodes will not be able to connect to the network without activation of this option.

The example of the node configuration file is represented below. This file does not include such options like *anchoring*, *Docker* smart contracts and private data access *groups*. Also enabling of the `sender` role in the `genesis` block is demonstrated.

Attention: Генезис это первый блок сети, от которого формируется блокчейн. Для включения роли `sender` необходимо явно указать версию генезиса (2), а также добавить сам параметр включения роли. Описание роли `sender` см. в разделе *Управление полномочиями*.

The ``apikey` key string hash authorization, as well as the Waves cryptography are also demonstrated. Description of the node configuration file parameters is available *here*.

Note: If you want to use additional options, set the `enable` field of the selected option to `yes` or `true` and configure the option section according to the description of its setting.

Warning: Please, fill **ONLY** the fields with the /**FILL**/ word inside as a value.

```
node {
# Type of cryptography
waves-crypto = yes

# Node owner address
owner-address = " /FILL/ "

# NTP settings
ntp.fatal-timeout = 5 minutes

# Node "home" and data directories to store the state
directory = "/node"
data-directory = "/node/data"

# Location and name of a license file
# license.file = ${node.directory}"/node.license"

wallet {
  # Path to keystore.
  file = "/node/keystore.dat"

  # Access password
  password = " /FILL/ "
}

# Blockchain settings
blockchain {
  type = CUSTOM
  fees.enabled = false
  consensus {
    type = "poa"
    round-duration = "17s"
    sync-duration = "3s"
    ban-duration-blocks = 100
    warnings-for-ban = 3
```

(continues on next page)

(continued from previous page)

```
    max-bans-percentage = 40
  }
  custom {
    address-scheme-character = "E"
    functionality {
      feature-check-blocks-period = 1500
      blocks-for-feature-activation = 1000
      pre-activated-features = { 2 = 0, 3 = 0, 4 = 0, 5 = 0, 6 = 0, 7 = 0, 9 = 0, 10 = 0, 100 = 0,␣
↪101 = 0 }
    }

    # Mainnet genesis settings
    genesis {
      version: 2
      sender-role-enabled: true
      average-block-delay: 60s
      initial-base-target: 153722867

      # Filled by GenesisBlockGenerator
      block-timestamp: 1573472578702

      initial-balance: 16250000 WEST

      # Filled by GenesisBlockGenerator
      genesis-public-key-base-58: ""

      # Filled by GenesisBlockGenerator
      signature: ""

      transactions = [
          # Initial token distribution:
          # - recipient: target's blockchain address (base58 string)
          # - amount: amount of tokens, multiplied by 10e8 (integer)
          #
          #     Example: { recipient: "3HQSr3VFCiE6JcWwV1yX8xttYbAGKTLV3Gz", amount: 30000000 WEST␣
↪}
          #
          # Note:
          #   Sum of amounts must be equal to initial-balance above.
          #
        { recipient: " /FILL/ ", amount: 1000000 WEST },
        { recipient: " /FILL/ ", amount: 1500000 WEST },
        { recipient: " /FILL/ ", amount: 500000 WEST },
      ]
      network-participants = [
          # Initial participants and role distribution
          # - public-key: participant's base58 encoded public key;
          # - roles: list of roles to be granted;
          #
          #     Example: {public-key: "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt", roles:␣
↪[permissioner, miner, connection_manager, contract_developer, issuer]}
          #
          # Note:
          #   There has to be at least one miner, one permissioner and one connection_manager for␣
↪the network to start correctly.
          #   Participants are granted access to the network via GenesisRegisterNodeTransaction.
```

(continues on next page)

(continued from previous page)

```
          #   Role list could be empty, then given public-key will only be granted access to the␣
↪network.
          #
        { public-key: " /FILL/ ", roles: [permissioner, sender, miner, connection_manager,␣
↪contract_developer, issuer]},
        { public-key: " /FILL/ ", roles: [miner, sender]},
        { public-key: " /FILL/ ", roles: []},
      ]
    }
  }
}

# Application logging level. Could be DEBUG | INFO | WARN | ERROR. Default value is INFO.
logging-level = DEBUG

tls {
  # Supported TLS types:
  # • EMBEDDED: Certificate is signed by node's provider and packed into JKS Keystore. The same␣
↪file is used as a Truststore.
  #             Has to be manually imported into system by user to avoid certificate warnings.
  # • DISABLED: TLS is fully disabled
  type = DISABLED

  # type = EMBEDDED
  # keystore-path = ${node.directory}"/we_tls.jks"
  # keystore-password = ${TLS_KEYSTORE_PASSWORD}
  # private-key-password = ${TLS_PRIVATE_KEY_PASSWORD}
}

# P2P Network settings
network {
  # Network address
  bind-address = "0.0.0.0"
  # Port number
  port = 6864
  # Enable/disable network TLS
  tls = no

  # Peers network addresses and ports
  #   Example: known-peers = ["node-1.com:6864", "node-2.com:6864"]
  known-peers = [  /FILL/  ]

  # Node name to send during handshake. Comment this string out to set random node name.
  #   Example: node-name = "your-we-node-name"
  node-name = " /FILL/ "

  # How long the information about peer stays in database after the last communication with it
  peers-data-residence-time = 2h

  # String with IP address and port to send as external address during handshake. Could be set␣
↪automatically if uPnP is enabled.
  #   Example: declared-address = "your-node-address.com:6864"
  declared-address = "0.0.0.0:6864"

  # Delay between attempts to connect to a peer
  attempt-connection-delay = 5s
```

(continues on next page)

(continued from previous page)

```
}

# New blocks generator settings
miner {
  enable = yes
  # Important: use quorum = 0 only for testing purposes, while running a single-node network;
  # In other cases always set quorum > 0
  quorum = 0
  interval-after-last-block-then-generation-is-allowed = 10d
  micro-block-interval = 5s
  min-micro-block-age = 3s
  max-transactions-in-micro-block = 500
  minimal-block-generation-offset = 200ms
}

# Nodes REST API settings
api {
  rest {
    # Enable/disable REST API
    enable = yes

    # Network address to bind to
    bind-address = "0.0.0.0"

    # Port to listen to REST API requests
    port = 6862

    # Enable/disable TLS for REST
    tls = no
  }

  grpc {
    # Enable/disable gRPC API
    enable = yes

    # Network address to bind to
    bind-address = "0.0.0.0"

    # Port to listen to gRPC API requests
    port = 6865

    # Enable/disable TLS for gRPC
    tls = no

    akka-http-settings {
      akka {
        http.server.idle-timeout = infinite
      }
    }
  }


  auth {
    type: "api-key"

    # Hash of API key string
```

(continues on next page)

(continued from previous page)

```
    # You can obtain hashes by running ApiKeyHash generator
    api-key-hash: " /FILL/ "

    # Hash of API key string for PrivacyApi routes
    privacy-api-key-hash: " /FILL/ "
  }
}

#Settings for Privacy Data Exchange
privacy {
  # Max parallel data crawling tasks
  crawling-parallelism = 100

  storage {
    vendor = none

    # for postgres vendor:
                        # schema = "public"
                        # migration-dir = "db/migration"
                        # profile = "slick.jdbc.PostgresProfile$"
                        # jdbc-config {
                        #   url = "jdbc:postgresql://postgres:5432/node-1"
                        #   driver = "org.postgresql.Driver"
                        #   user = postgres
                        #   password = wenterprise
                        #   connectionPool = HikariCP
                        #   connectionTimeout = 5000
                        #   connectionTestQuery = "SELECT 1"
                        #   queueSize = 10000
                        #   numThreads = 20
                        # }

                        # for s3 vendor:
                        # url = "http://localhost:9000/"
                        # bucket = "privacy"
                        # region = "aws-global"
                        # access-key-id = "minio"
                        # secret-access-key = "minio123"
                        # path-style-access-enabled = true
                        # connection-timeout = 30s
                        # connection-acquisition-timeout = 10s
                        # max-concurrency = 200
                        # read-timeout = 0s
  }

  cleaner {
    enabled: no

    # The amount of time between cleanups
    # interval: 10m

    # How many blocks the data hash transaction exists on the blockchain, after which it will be␣
↪removed from cleaner monitoring
    # confirmation-blocks: 100

    # The maximum amount of time that a file can be stored without getting into the blockchain
```

(continues on next page)

(continued from previous page)

```
    # pending-time: 72h
  }
}


# Docker smart contracts settings
docker-engine {
  # Docker smart contracts enabled flag
  enable = yes

  # For starting contracts in a local docker
  use-node-docker-host = yes

  default-registry-domain = "registry.wavesenterprise.com/waves-enterprise-public"
  # Basic auth credentials for docker host
   #docker-auth {
   #  username = "some user"
   #  password = "some password"
   #}

  # Optional connection string to docker host
   docker-host = "unix:///var/run/docker.sock"

  # Optional string to node REST API if we use remote docker host
  # node-rest-api = "node-0"

  # Execution settings
  execution-limits {
    # Contract execution timeout
    timeout = 10s
    # Memory limit in Megabytes
    memory = 512
    # Memory swap value in Megabytes (see https://docs.docker.com/config/containers/resource_
↪constraints/)
    memory-swap = 0
  }

  # Reuse once created container on subsequent executions
  reuse-containers = yes

  # Remove container with contract after specified duration passed
  remove-container-after = 10m

  # Remote registries auth information
  remote-registries = []

  # Check registry auth on node startup
  check-registry-auth-on-startup = yes

  # Contract execution messages cache settings
  contract-execution-messages-cache {
    # Time to expire for messages in cache
    expire-after = 60m
    # Max number of messages in buffer. When the limit is reached, the node processes all messages␣
↪in batch
    max-buffer-size = 10
```

(continues on next page)

(continued from previous page)

```
    # Max time for buffer. When time is out, the node processes all messages in batch
    max-buffer-time = 100ms
  }
}
}
```

18.2 Changes in the node configuration file

This section provides information to help you identify changes in the configuration file depending on the node version.

Warning: If you are updating a node version, you must also update the node configuration file. The node will not run without updating the configuration file!

18.2.1 Changes in the node configuration file of the 1.5.0 version

Due to realization of the *CFT* consensus algorithm and the *sender* role, the following sections of the node configuration file have been changed:

Blockchain section

The `cft` consensus type has been added for the `consensus` block, as well as two parameters needed for validation of blocks:

- `maxvalidators` maximal number of validators defined for a definite voting round.
- `finalizationtimeout` time period, during which a miner waits for finalization of the last block (in seconds).

Genesis section

- For explicit definition of a genesis version being in use, the `version` parameter has been added. The version 1 is configured by default, the version 2 is used for enabling of the `sender` role.
- The `senderroleenabled` parameter has been added for enabling of the `sender` role. In order to enable the role, type `true` as the parameter value, for desabling type `false`.

Attention: The `sender` role is defined in the genesis block. That is why it will work only for new networks built on the platform version not earlier than 1.5.0.

18.2.2 Changes in the node configuration file of the 1.4.0 version

The database for storing the state inside the node has changed, now RockDB is used instead of LevelDB. If you upgrade to release 1.4.0, you can't roll back. Database migration occurs automatically when switching to release 1.4.0 and may take a long time. If you have several nodes in the network, then the nodes must be updated strictly sequentially! It is recommended to make a backup copy of the old database.

api section

The `restapi` section was renamed to the `api` section. The section now includes *settings* for the REST API and gRPC interface.

privacy section

The following changes were added to the section:

- The `cleaner` block was added
- S3 Minio option was added for the data storage

18.2.3 Changes in the node configuration file of the 1.2.2 version

Blockchain section

The **Mainnet** `blockchain` section needs to be changed instead of full settings to this one:

```
blockchain.type = MAINNET
```

> **Warning:** If the node, which is connected to the Mainnet, has old `blockchain` section settings, it will fork!

The `blockchain` section corresponds to individual settings in all other cases.

18.2.4 Changes in the node configuration file of the 1.2.0 version

dockerengine section

In the section `dockerengine` added parameter `grpcserver`, responsible for setting up gRPC server to work docker contracts with gRPC API:

```
grpc-server {
  # gRPC server port
  port = 6865
  # Optional node host
  # host = "192.168.65.2"
}
```

18.2.5 Changes in the node configuration file for earlier versions

Node version 1.1.2

Node version 1.1.0

18.3 Description of the node configuration file parameters and sections

Several types of values are used for parameters in the configuration file:

- Integer data which used to specify the exact number of elements. It can be the number of transactions, blocks or connections.
- Integer data including measuring units to specify the time periods or memory volume. You typically specify the time periods in days, hours, or seconds, or the cache memory volume, for example, `memory = 256M` or `connectiontimeout = 30s`.

- String which used to specify the addresses, directory paths, passwords and so on. The directory path is specifying in the acceptable format of your current OS and the value is quoted.
- Array for the list of values like addresses or public keys. The value is specified in square brackets separated by commas.
- Boolean `no` or `yes` which used for option activation.

An example of the node configuration file is represented on the *configuration files prepare* page. It includes the following sections:

- *node* general section, which includes all sections of blockchain settings.
- *synchronization.transactionbroadcaster* synchronization parameters settings for sending unconfirmed transactions to the blockchain.
- *additionalcache* configuring additional cache memory settings for temporary storage of incoming blocks.
- *loggers* detailed configuration of loggers.
- *ntp* NTP server parameters settings.
- *blockchain* common blockchain settings.
- *features* network settings.
- *tls* enabling and setting up of the node TLS.
- *network* network settings.
- *wallet* settings of the private keys access.
- *miner* mining settings.
- *restapi* REST API/gRPC settings and authorization type.
- *privacy* confidential information access groups settings.
- *dockerengine* Docker smart contracts settings.

18.3.1 `node` section

Additional section parameters:

- `wavescrypto` *cryptography* type in the blockchain. Possible values: `yes` Waves cryptography, `no` GOST cryptography.
- `directory` the main directory for the storage of the node software.
- `datadirectory` a directory for storing blockchain data in RocksDB: blocks, transactions, state nodes.
- `logginglevel` logging level. Possible values: `DEBUG`, `INFO`, `WARN`, `ERROR`, default value is `INFO`.
- `owneraddress` the node address, the future owner of the configuration file.

18.3.2 synchronization.transactionbroadcaster section

- maxbatchsize and maxbatchtime – technical parameters that allow you to adjust the speed of reducing the transaction queue.
- minbroadcastcount – a minimum number of connections that can be used to send each transaction to the blockchain. The value should not exceed the number of nodes in the network minus one (the sender should not be taken into account).
- retrydelay – an interval for resending a transaction if the number of current connections was not enough, or errors occurred during sending.
- extensionbatchsize a number of blocks in the series used to request an extension from peers.
- knowntxcachesize the maximum number of unconfirmed transactions in the cache.
- knowntxcachetime the maximum lifetime of unconfirmed transactions in the memory cache.

18.3.3 additionalcache section

- rocksdb RocksDB database parameters:
  - maxcachesize the maximum cache size.
  - maxrollbackdepth the number of blocks that the node can be manually rolled back.
  - rememberblocksintervalincache a storing period for blocks in the cache.
- blockids cache parameters for incoming blocks.
  - maxsize maximum size of the cache.
  - expireafter a period, after which stored blocks will be deleted.

18.3.4 loggers section

This section is intended for listing loggers with an individually set of a logging level. You can find out the list of loggers by using the *GET /node/logging* method. Loggers are specified as follows:

```
"com.wavesplatform.mining.MinerImplPoa": TRACE
    "com.wavesplatform.utx.UtxPoolImpl": DEBUG
```

18.3.5 ntp section

- server an NTP server addresses list. The recommended value is [«0.pool.ntp.org», «1.pool.ntp.org», ... «10.pool.ntp.org»].
- requesttimeout the timeout of the one request to an NTP server. The recommended value is 10 seconds.
- expirationtimeout the timeout of the NTP server requests synchronization. The recommended value is 1 minute.
- fataltimeout the timeout of the connection to an NTP server. The recommended value is 1 minute.

18.3.6 `blockchain` section

- `type` the blockchain type. Possible values are `MAINNET` or `CUSTOM`. The `MAINNET` value allows you to use the genesis block, consensus and Mainnet settings. When you select `MAINNET` in the configuration file of the node which connects to the Mainnet network, you do not need to specify the parameters of `custom`, `genesis` and `consensus` blocks.
- `consensus.type` *consensus* type. Possible values are `pos`, `poa` or `cft`. You can read more *here* about consensus settings.

`fees` **unit**

- `enabled` the option of using fees for the *transaction* release. Possible values are `false` or `true`.

`custom` **unit**

- `addressschemecharacter` the address feature character which is used to prevent mixing up addresses from different networks. For the "Waves Enterprise Mainnet" `V` and for the "Waves Enterprise Partnernet" `P`. You can use any letter you like for the sidechain or test versions of the Waves Enterprise blockchain platform. Nodes must have the same network byte on the same blockchain network.
- `functionality` main blockchain settings.
- `genesis` genesis block settings.

`functionality` **unit**

- `featurecheckblocksperiod` the blocks period for feature checking and activation.
- `blocksforfeatureactivation` the number of blocks required to accept feature.
- `preactivatedfeatures` a set of blockchain options.

`genesis` **unit**

- `averageblockdelay` an average delay between the blocks creation. This parameter is used only for the *PoS* consensus.
- `initialbasetarget` an initial base number for the managing the mining process. This parameter is used for the *PoS* consensus . The frequency of the block creation depends on the parameter value therefore the higher the value, the more often blocks are created. Also, the value of the miner's balance affects the use of this parameter in mining the larger the miner's balance, the less the value of `initialbasetarget` is used. When setting a value for this parameter, it is recommended to take into account the combination of miners balances and the expected interval between blocks.
- `blocktimestamp` a time and data code. The time is specified in milliseconds and the value must consist of 13 digits. If you specify the standard value `timestamp` consisting of 10 digits, then you need to add any three digits at the end.
- `initialbalance` an initial balance in smallest units. The parameter value affects on the mining process with the *PoS* consensus. The larger the miner's balance, the smaller the `initialbasetarget` value is used for the mining node determination for the current round.
- `genesispublickeybase58` the public key hash of the genesis block, encrypted in Base58.
- `signature` the genesis block signature, encrypted in Base58.
- `transactions` a list of network participants with an initial balance, the creation of which will be included in the genesis block.
- `networkparticipants` a list of network participants with specified roles, the creation of which will be included in the genesis block.

18.3.7 `tls` section

This section is dedicated to the node TLS.

- `type` TLS mode status. Possible options: `DISABLED` (disabled, in this case other options should be excluded or commented) and `EMBEDDED` (enabled, the certificate is signed by a node provider and packed within a JKS file (keystore); the certificate directory and keystore access parameters should be stated by a user in the fields below).
- `keystorepath` keystore relative path within the node directory: `${node.directory}"/we_tls.jks"`.
- `keystorepassword` the password for the generated keystore.
- `password` a password for the private keys file access.

In order to work with the node TLS, apart its configuration in the node config file, a user should get a keystore file itself with the use of the **keytool** utility:

```
keytool \
 -keystore we.jks -storepass 123456 -keypass 123456 \
 -genkey -alias we -keyalg RSA -validity 9999 \
 -dname "CN=Waves Enterprise,OU=security,O=WE,C=RU" \
 -ext "SAN=DNS:welocal.dev,DNS:localhost,IP:51.210.211.61,IP:127.0.0.1"
```

- `keystore` keystore file name.
- `storepass` keystore password, which should be stated in the `keystorepassword` section of the node config file.
- `keypass` private key password, which should be stated in the `privatekeypassword` section of the config file.
- `alias` an alias name (upon a user decision).
- `keyalg` keypair generation algorithm.
- `validity` keypair validity time in days.
- `dname` distinguished name according to the X.500 standard, connected with the keystore alias.
- `ext` extensions that are used for key generation, all possible host names and IP addresses should be stated for work in different networks.

As a result of the keytool utility execution, the keystore file with the filename **we.jks** will be obtained. In order to connect with the node operating with the TLS, a user should also generate a client certificate:

```
keytool -export -keystore we.jks -alias we -file we.cert
```

The obtained certificate file **we.cert** should be imported into the trusted certificate storage. If the node is located in one network with a user, it will be enough to state a relative path to the we.jks file in the node config file, as demonstrated above.

In case the node is located in another network, a we.cert certificate file should be imported into the keystore:

```
keytool -importcert -alias we -file we.cert -keystore we.jks
```

After that, state the relative path to the we.jks file in the node config file.

18.3.8 `network` section

- `bindaddress` the node network address.
- `port` the port number.
- `knownpeers` a list of known nodes network addresses. This parameter should be filled in. The list of addresses is passed to the user by the network administrator before the new node is connected.
- `declaredaddress` a string with IP address and port to send as external address during the handshake.
- `maxsimultaneousconnections` a maximum number of simultaneously supported connections. This parameter is limited by the number of nodes in the blockchain, i.e. the maximum number of simultaneous connections will not exceed the number of nodes in the network.
- `peersrequestinterval` an interval for requesting a list of peers. The value is specified in seconds or minutes. The recommended value is 12 minutes.
- `attemptconnectiondelay` a request interval to connect to any of the known peers.

18.3.9 `wallet` section

- `file` a path to the private keys storage.
- `password` a password for the private keys file access.

18.3.10 `miner` section

- `enable` a miner option activation.
- `quorum` required number of connections (both incoming and outgoing) to attempt block generation. Setting this value to `0` enables offline generation. When you are specifying the value, it is necessary to consider that the own mining node is not summed with the parameter value, i.e., if it is `quorum = 2`, then you need at least 3 mining nodes in the network.
- `intervalafterlastblockthengenerationisallowed` enable block generation only if the last block is not older the given period of time.
- `microblockinterval` an interval between microblocks.
- `minmicroblockage` a minimal age of the microblock.
- `maxtransactionsinmicroblock` a maximum number of transaction in the microblock.
- `minimalblockgenerationoffset` a minimal time interval between blocks.

18.3.11 `features` section

- `supported` a list of supported options.

18.4 Accounts creation

The user account includes an address and a key pair which consists of public and private keys. The address and public key are shown to the user during account creation on the command line. The private key is written to the `keystore.dat`.

18.4.1 Key pairs generating

Public and private keys for initial participants are creating by the generator. You can get the last version of the generator on our GitHub page. Before running the utility you need to specify the `accounts.conf` configuration file which contains parameters for keys creating. During the creation think up and enter a password, then save it for later configuration. The given password will be used at creation of a global variable `WE_NODE_OWNER_PASSWORD` further. Press `enter` key if you do not want to use this password. Use the following command to run the generator:

```
java jar generatorsx.x.x.jar AccountsGeneratorApp accounts.conf
```

18.4.2 Global variables

We recommend to use a password for the keys pair to increase security. The Waves Enterprise platform supports two ways of the password usage:

1. Enter the password manually at the each start of the node.
2. Create global variables in your OS.

If you are using the manual enter the password there is no need to create global variables. But when you are planning to use containers or any similar services to run the node then create the following global variables in the OS for your convenience:

1. `WE_NODE_OWNER_PASSWORD` the keys pair password specified during the key pair creation.
2. `WE_NODE_OWNER_PASSWORD_EMPTY` `true` or `false`, specify the `true` value if you do not want to use the keys pair password, in this case it is not necessary to create the `WE_NODE_OWNER_PASSWORD` variable. When you are using the password than specify the `false` value and write into the `WE_NODE_OWNER_PASSWORD` variable the keys pair password.

18.5 Signing the genesis block

Sign the genesis block using utility generatorsx.x.x.jar. Command for signing: `java jar generatorsx.x.x.jar GenesisBlockGenerator node.conf`, where `Name.conf` is the edited in *this section* node configuration file. After signing `genesispublickeybase58` and `signature` fields of the configuration file will be filled with values of the public key and the proof of the genesis block.

Example:

```
genesis-public-key-base-58: "4ozcAj...penxrm"
signature: "5QNVGF...7Bj4Pc"
```

18.6 Consensus settings

Блокчейнплатформа Waves Enterprise поддерживает три типа консенсуса *PoS*, *PoA* и *CFT*. Настройки консенсуса располагаются в секции *blockchain*.

18.6.1 PoS configuration

The PoS consensus will be used by default if you have not specified the consensus type in the `consensus.type` field of the *blockchain* section. Here are the mining responsible parameters which are located in the `genesis` unit of the *blockchain* section:

- `averageblockdelay` an average delay between the blocks creation. The default value is 60 seconds. The value of this parameter is ignored if PoA consensus is selected.
- `initialbasetarget` an initial base number for the managing the mining process. The frequency of the block creation depends on the parameter value therefore the higher the value, the more often blocks are created. Also, the value of the miner's balance affects the use of this parameter in mining the larger the miner's balance, the less the value of `initialbasetarget` is used.
- `initialbalance` an initial balance in smallest units. The greater the share of the miner's balance from the network initial balance, the smaller becomes the value of `initialbasetarget` to determine the node miner of the current round.

We recommend to use the default parameter values specified in the configuration files examples which are represented on the GitHub page.

18.6.2 PoA settings

Please, uncomment or add the `consensus` unit of the *blockchain* section for the *PoA* consensus usage:

```
consensus {
  type = "poa"
  round-duration = "17s"
  sync-duration = "3s"
  ban-duration-blocks = 100
  warnings-for-ban = 3
  max-bans-percentage = 40
   }
```

The parameters represtnted in the `consensus` unit are used only for the *PoA* consensus.

- `type` the consensus type. Possible values are `pos`, `poa` or `cft`. If the `pos` value is specified, other parameters will not be considered. The CFT consensus is described *below*.
- `roundduration` a round length of the block mining in seconds.
- `syncduration` a block mining synchronization period in seconds. The total time of the round is the sum of `roundduration` and `syncduration`.
- `bandurationblocks` a blocks quantity of the ban period for the mining node.
- `warningsforban` a number of rounds which is used for ban warnings for miner nodes.
- `maxbanspercentage` a percentage of mining nodes from the total number of nodes in the network that can be placed in the ban.

Using the *PoA* consensus allows to adjust the order of blocks creation by limiting the mining function for certain nodes. The reason is to distribute evenly the network load, if any mining nodes left the network or became inactive. Mining node can get banned for the following reasons:

- if a node will miss its queue for mining;
- if a node provides an invalid block;
- if a node went offline.

Before getting into the `blacklist` the mining node receives warnings about the ban possibility during the number of rounds that is specified in the `warningsforban` parameter. The mining node will be back to the mining after the `bandurationblocks` parameter value will end.

18.6.3 CFT configuration

Для использования консенсуса *CFT* также необходимо раскомментировать или добавить блок `consensus` в секции *blockchain*: Основные параметры настройки CFT идентичны параметрам консенсуса *PoA*:

```
consensus {
  type: "cft"
  round-duration: 7s
  sync-duration: 3s
  ban-duration-blocks: 14
  warnings-for-ban: 2
  max-bans-percentage: 33
  max-validators: 7
  finalization-timeout: 2s
  }
```

However, in comparison with the PoA consensus, CFT uses two additional configuration parameters needed for validation of blocks during a voting round:

- `maxvalidators` – limit of validators taking part in a definite round.
- `finalizationtimeout` – time period, during which a miner waits for finalization of the last block in the chain. After this time, a miner returns transactions to the UTX pool and starts to main a round anew.

18.6.4 Consensus settings in the `miner` section

When you are configuring consensus settings, please, consider the following settings of the *miner* section:

- `microblockinterval` an interval between microblocks. The value is specified in seconds.
- `minmicroblockage` a minimal age of the microblock. The value is specified in seconds and should not be more than the `microblockinterval` parameter value.
- `minimalblockgenerationoffset` a minimal time interval between blocks. The value is specified in milliseconds.

The values of the microblock creation parameters should not conflict with the parameters values of the `averageblockdelay` for PoS and `roundduration` for PoA and CFT. The number of microblocks in a block is not limited, but depends on the transactions size in the microblock.

18.7 Docker configuration

Installation and execution of docker smart contracts is configured in the `dockerengine` of the *node configuration file*.

See details of configuration of Docker smart contracts in the *Preparing to work* section of the *Socker smart contracts* chapter.

18.8 Authorization type configuration for the REST API and gRPC access

The Waves Enterprise blockchain platform supports the following two types of authorization for the node's REST API/gRPC access:

- `apikey` string hash authorization;
- authorization using the JWT token.

`apikey` string hash authorization type is a simple method of the access management to a node with a low level security. If the `apikey` hash is leaking out to the attacker, he is getting the full access to the node. When you utilize the separate authorization service with access tokens, you increase the security level of your blockchain network to the high level. You can read more information about the authorization service in the *Authorization service* section.

18.8.1 `api` section of the node configuration file

`api` section contains authorization settings and REST API/gRPC interfaces.

```
api {
rest {
  # Enable/disable REST API
  enable = yes

  # Network address to bind to
  bind-address = "0.0.0.0"

  # Port to listen to REST API requests
  port = 6862

  # Enable/disable TLS for REST
  tls = no

  # Enable/disable CORS support
  cors = yes

  # Max number of transactions
  # returned by /transactions/address/{address}/limit/{limit}
  transactions-by-address-limit = 10000

  distribution-address-limit = 1000
}

grpc {
  # Enable/disable gRPC API
```

(continues on next page)

(continued from previous page)

```
  enable = yes

  # Network address to bind to
  bind-address = "0.0.0.0"

  # Port to listen to gRPC API requests
  port = 6865

  # Enable/disable TLS for GRPC
  tls = no

  # Akka HTTP settings for gRPC server
  akka-http-settings {
    akka {
      http.server.idle-timeout = infinite

      # Uncomment these settings if you want detailed logging for gRPC calls
      # loggers = ["akka.event.slf4j.Slf4jLogger"]
      # loglevel = "DEBUG"
      # logging-filter = "akka.event.slf4j.Slf4jLoggingFilter"
      # stdout-loglevel = "DEBUG"
      # log-dead-letters = 10
      # log-dead-letters-during-shutdown = on
      #
      # actor {
      #   debug {
      #     # enable function of LoggingReceive, which is to log any received message at
      #     # DEBUG level
      #     receive = on
      #     # enable DEBUG logging of all AutoReceiveMessages (Kill, PoisonPill etc.)
      #     autoreceive = on
      #     # enable DEBUG logging of actor lifecycle changes
      #     lifecycle = on
      #     # enable DEBUG logging of unhandled messages
      #     unhandled = on
      #     # enable DEBUG logging of subscription changes on the eventStream
      #     event-stream = on
      #     # enable DEBUG logging of all LoggingFSMs for events, transitions and timers
      #     fsm = on
      #   }
      # }
      #
      # io.tcp.trace-logging = on
      # http.server.http2.log-frames = yes
    }
  }
}

# Authorization strategy should be either 'oauth2' or 'api-key', default is 'api-key'
auth {
  type = "api-key"

  # Hash of API key string
  api-key-hash = "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"

  # Hash of API key string for PrivacyApi routes
```

(continues on next page)

(continued from previous page)

```
  privacy-api-key-hash = "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}
# For OAuth2:
# auth {
#   type: "oauth2"

#   # OAuth2 service public key to verify auth tokens
#   public-key: "AuthorizationServicePublicKeyInBase64"

# }
}
```

api.rest parameters description

- `enable` REST API option activation.
- `bindaddress` a network address to bind the REST API interface.
- `port` a port to listen to REST API requests.
- `tls` enable/disable TLS for REST API requests.
- `cors` enable/disable CORS support.
- `transactionsbyaddresslimit` a maximum number of transactions returned by `/transactions/address/{address}/limit/{limit}` method.
- `distributionaddresslimit` `GET /assets/{assetId}/distribution/{height}/limit/{limit}`.

api.grpc parameters description

- `enable` gRPC interface activation.
- `bindaddress` a network address to bind the gRPC interface.
- `port` a port to listen to gRPC requests.
- `tls` enable/disable TLS for gRPC requests.

`auth` section for the `apikey` type

- `type` the authorization type, specify the `apikey` value the string hash authorization.
- `apikeyhash` a hash of API key string.
- `privacyapikeyhash` a hash of API key string for `privacy` methods.

`auth` section for the `oauth2` type

- `type` the authorization type, specify the `oauth2` value the token authorization.
- `publickey` a public key of the authorization service.

The REST API and gRPC interfaces use the same `apikey` and JWTtoken values.

18.8.2 Key string authorization usage

Specify the `apikey` value for the `authtype` parameter. Create the `apikeyhash` for the REST API access by using the generatorsx.x.x.jar utility. To run the utility, you need to specify the `apikeyhash.conf` file as one of the parameters, which defines the parameters of creating the `apikeyhash`. Use the following command to run the generator:

```
java -jar generators-x.x.x.jar ApiKeyHash api-key-hash.conf
```

Specify the value obtained as a result of the utility execution in the parameter `apikeyhash` in the node configuration file.

Create the `privacyapikeyhash` by the same way as the `apikeyhash` to get the *privacy* methods access. Specify the value obtained as a result of the utility execution in the parameter `privacyapikeyhash` in the node configuration file.

18.8.3 Token authorization usage

Specify the `oauth2` value for the `authtype` parameter, write the public key of the authorization service into the `publickey` parameter.

18.9 Anchoring settings

If you are using the *anchoring* option, please, configure the `anchoring` unit. `targetnet` is the blockchain network which will be used by the sidechain node to send anchoring transactions.

```
anchoring {
enable = yes
height-range = 30
height-above = 8
threshold = 20
tx-mining-check-delay = 5 seconds
tx-mining-check-count = 20


targetnet-authorization {
type = "oauth2" # "api-key" or "oauth2"
authorization-token = ""
authorization-service-url = "https://client.wavesenterprise.com/authServiceAddress/v1/
↪auth/token"
token-update-interval = "60s"
# api-key-hash = ""
# privacy-api-key-hash = ""
}

targetnet-scheme-byte = "V"
targetnet-node-address = "https://client.wavesenterprise.com:6862/NodeAddress"
targetnet-node-recipient-address = ""
targetnet-private-key-password = ""

 wallet {
 file = "node-1_mainnet-wallet.dat"
 password = "small"
  }
```

(continues on next page)

(continued from previous page)

```
 targetnet-fee = 10000000
 sidechain-fee = 5000000
}
```

Anchoring parameters

- `heightrange` the number of blocks which is used as an interval between anchoring transactions to the Targetnet.
- `heightabove` the number of blocks in the Targetnet after which the private blockchain node creates the confirming datatransaction containing data from the first datatransaction. We recommend specifying this value that does not exceed the Targetnet maximum rollback depth `maxrollback`.
- `threshold` the number of blocks subtracted from the current height of the private blockchain. The anchoring transaction sent to the Targetnet includes the data from the block at height `currentheight threshold`. When the value is 0, the current block is anchored. We recommend specifying this value close to the private blockchain maximum rollback depth `maxrollback`.
- `txminingcheckdelay` время ожидания между проверками доступности транзакции для анкоринга в Targetnet.
- `txminingcheckcount` максимальное количество проверок доступности транзакции для анкоринга в Targetnet, по выполнении которых транзакция считается не поступившей в сеть.

The distance between anchoring transactions may change depending on the mining settings in the `Targetnet` network. The specified value `heightrange` sets the approximate interval between anchoring transactions. The real time of falling anchoring transactions into the mined block of the `Targetnet` may exceed the time spent on the mining of the `heightrange` number of blocks.

Anchoring authorization parameters

- `type` authorization type for anchoring. `apikey` `apikeyhash` authorization , `authservice` authorization by a special security token.

For authorization by `apikeyhash` necessary a current keyvalue as `apikey`. For authorization by a special security token you must use a `type = "authservice"` and comment configfile structure values:

- `authorizationtoken` a constant authorization token.
- `authorizationserviceurl` URL address authorization service.
- `tokenupdateinterval` data interval for a token refresh.

Targetnet access parameters

A separate `keystore.dat` file with a key pair for the Targetnet access is generated for the node that will send the anchoring transaction to the Targetnet.

- `targetnetschemebyte` the Targetnet network byte.
- `targetnetnodeaddress` the full node network address including the port number in the Targetnet for the sending of anchoring transactions. The address should be specified along with the connection type (http/https), the port number and the `NodeAddress` parameter as in the example `http://node.weservices.com:6862/NodeAddress`.
- `targetnetnoderecipientaddress` the node address in the Targetnet for the recording of anchoring transactions signed with a key pair of this address.
- `targetnetprivatekeypassword` the node private key password for the anchoring transactions signing.

The network address and the port for the Targetnet/Partnernet networks anchoring can be obtained from Waves Enterprise technical support staff. If multiple private blockchains with mutual anchoring are used, you should use the appropriate private network settings.

Parameters of key pair file for the Targetnet anchoring transactions signing, wallet unit

- `file` a file name and a path to the key pair file for the Targetnet anchoring transactions signing. The file is located on the private network node.
- `password` a password of the key pair file.

Fee parameters

- `targetnetfee` the fee for the anchoring transaction issue in the Targetnet.
- `sidechainfee` the fee for the anchoring transaction issue in the private blockchain.

18.10 Privacy data access groups configuration

When using the *privacy* methods activate the option and fill in the `storage` block with database settings for storing the private data:

```
privacy {
# Max parallel data crawling tasks
crawling-parallelism = 100

storage {
  vendor = none

  # for postgres vendor:
                    # schema = "public"
                    # migration-dir = "db/migration"
                    # profile = "slick.jdbc.PostgresProfile$"
                    # jdbc-config {
                    #   url = "jdbc:postgresql://postgres:5432/node-1"
                    #   driver = "org.postgresql.Driver"
                    #   user = postgres
                    #   password = wenterprise
                    #   connectionPool = HikariCP
                    #   connectionTimeout = 5000
                    #   connectionTestQuery = "SELECT 1"
                    #   queueSize = 10000
                    #   numThreads = 20
                    # }

                    # for s3 vendor:
                    # url = "http://localhost:9000/"
                    # bucket = "privacy"
                    # region = "aws-global"
                    # access-key-id = "minio"
                    # secret-access-key = "minio123"
                    # path-style-access-enabled = true
                    # connection-timeout = 30s
                    # connection-acquisition-timeout = 10s
                    # max-concurrency = 200
                    # read-timeout = 0s
}
```

(continues on next page)

(continued from previous page)

```
cleaner {
  enabled: no

  # The amount of time between cleanups
  # interval: 10m

  # How many blocks the data hash transaction exists on the blockchain, after which it␣
↪will be removed from cleaner monitoring
  # confirmation-blocks: 100

  # The maximum amount of time that a file can be stored without getting into the␣
↪blockchain
  # pending-time: 72h
}
}
```

Parameters description

- `vendor` selecting a data storage option: `s3` cloud or local storage based on Amazon Simple Storage Service (S3), `postgres` local storage based on PostgreSQL DB. A Minio server is used for the data storage.

PostgreSQL DB parameters:

- `url` the PostgreSQL DB address;
- `driver` the JDBC driver name;
- `profile` a profile name for the JDBC access;
- `user` a user name for the DB access;
- `password` a password for the DB access;
- `connectionPool` a connection pool name, default is `HikariCP`.
- `connectionTimeout` a connection timeout;
- `connectionTestQuery` a query name for the connection test;
- `queueSize` a requests queue size;
- `numThreads` a number of parallel connections;
- `schema` an interaction scheme;
- `migrationdir` a path to the data migration directory.

S3 parameters:

- `url` an address of the S3 server for data storage, Minio servers are supported;
- `bucket` a name of the S3 database table to store data;
- `region` the name of the S3 region, the parameter value is `awsglobal`;
- `accesskeyid` ID of the data access key;
- `secretaccesskey` key for accessing data in S3 storage;
- `pathstyleaccessenabled = true` immutable parameter for specifying the path to the S3 table;
- `connectiontimeout` a connection timeout;
- `connectionacquisitiontimeout` a timeout to get a connection;

- `maxconcurrency` a number of concurrent accesses to the storage;
- `readtimeout` data read timeout.

`cleaner` section

- `enabled` enable/disable periodic deletion of files that are not included in the blockchain;
- `interval` files cleaning interval;
- `confirmationblocks` the time period in blocks for which a hash transaction of data exists in the blockchain, after which it will be deleted;
- `pendingtime` the maximum period of time for which a file with data can be stored without getting into the blockchain.

Other parameters:

- `requesttimeout` a waiting timeout for all responses from peers to a data request.
- `initretrydelay` a delay from the receiving of the data hash to the start of its search among peers.
- `crawlingparallelism` a limitation of the maximum number of simultaneous processes in the synchronizer.
- `maxattemptcount` the maximum number of rounds for requesting data from peers after which data is considered "lost".
- `lostdataprocessingdelay` an interval of rounds of requests for "lost" data.
- `cache` responses cache settings.

DB PostgreSQL is using as a database for the confidential data storage. The database should be installed on the same machine with the node and should have an DB access account. You can use the PostgreSQL tutorial for download and install the database according with your operation system type.

During the installation the system will offer to create an access account. These credentials must be entered into the appropriate `user/password` parameters.

Specify the URL for the PostgreSQL connection into the `url` parameter. URL consists of:

- POSTGRES_ADDRESS a PostgreSQL host address;
- POSTGRES_PORT a PostgreSQL host port number;
- POSTGRES_DB a PostgreSQL name.

You can specify the PostgreSQL credentials with the URL in the same string. The example is represented bellow, where `user=user_privacy_node_0@wedev` is a login, `password=7nZL7Jr41qOWUHz5qKdypA&sslmode=require` a password with require option during the authorization.

Example

```
privacy.storage.url = "jdbc:postgresql://vostk-dev.postgres.database.azure.com:5432/
↪privacy_node_0?user=user_privacy_node_0@we-dev&password=7nZL7Jr41qOWUHz5qKdypA&
↪sslmode=require"
```

You can download the latest distributives and configuration files examples from the GitHub Waves Enterprise release page.

CHAPTER

NINETEEN

USING A LICENSE

The Waves Enterprise blockchain platform is commercial and is designed primarily for use in large companies and the public sector. To use the technology, you must purchase a license for the platform. Quick and easy access to the list of licenses is provided by the licensing service.

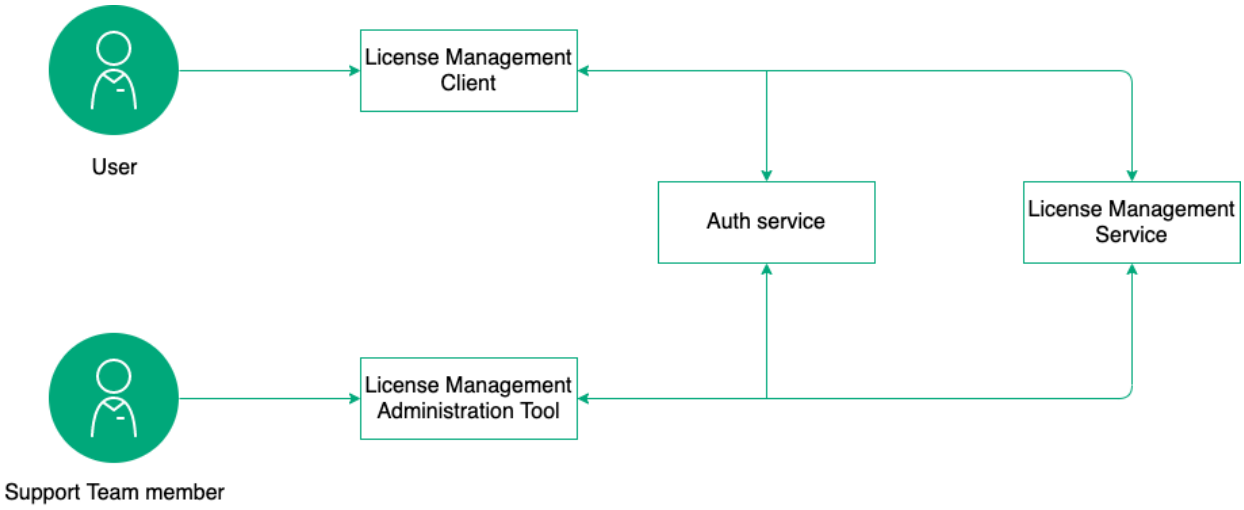


Fig. 1: Waves Enterprise blockchain platform license acquisition scheme

You do not need a product license to learn about the platform's features. The platform retains full functionality until the blockchain height of **30,000** blocks is reached, which at block round time of 30 seconds is 10 days of operation without restrictions.

Waves Enterprise blockchain platform users are offered the following license types:

- **Commercial license** allows you to use the platform to implement commercial projects. It is issued for the period determined by the contractual relations with the partner.
- **Noncommercial license** allows using the platform for implementing noncommercial projects. It is issued for the period determined by the contractual relations with the partner.
- **Trial license** allows you to familiarize yourself with the platform and the technology. It is issued for the duration of the pilot project by contract, or for the time of product development and debugging.
- **The Mainnet network license** is a special license that allows you to run the node in the *Mainnet* network. To work in the network you should have at least **50,000 WEST** on your balance or in leasing. If the specified balance is reduced, restrictions on block formation and access to the node API are introduced. Sending an application for registration of new members is performed in the Service Desk system.

Attention:	One license applies to one node!

Based on their validity periods, license types include:

- Indefinite.
- Twoyear.
- Oneyear.
- Threemonth (trial license).
- Rental for the period of use of the technology.

Upon license expiration, the node for which the license was purchased will no longer be able to generate blocks or write new transactions to the network.

19.1 Obtaining a license

To formalize a license request, follow these steps:

1. Go to license management service and create a new account, if it has not been created before.
2. Send your license request to Waves Enterprise support. A support representative will contact you to agree on the details, create a company profile, and link the created account to it.
3. After activating the license, specify the address of your node (*node_owner_address*).
4. Send the specified license file as JSON in the request *POST /licenses/upload* to the node.
5. To view the license status, use the request *GET /licenses/status*.

CHAPTER

TWENTY

MAINNET AND PARTNERNET CONNECTION

20.1 Working inside the "Waves Enterprise Mainnet"

20.1.1 Connection of the node to the "Waves Enterprise Mainnet"

Warning: The account balance must be at least **50,000 WEST** if you want to connect your node to the network "Waves Enterprise Mainnet" and do mining! Information about the generating balance in the Mainnet network is updated once every 1000 blocks, mining will only be available after the generating balance is updated.

Follow these steps for the node connection to the "Waves Enterprise Mainnet":

1. Go to the Waves Enterprise website and create an account following the webinterface hints.
2. Transfer tokens to the "Waves Enterprise Mainnet" network.
3. Transfer for leasing any number of tokens to the `3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu` address and keep the transaction ID. Further you can withdraw tokens from the lease, because this operation is necessary to verify your ownership of this address and the balance.
4. *Deploy* a single node.
5. Go to the Waves Enterprise support website and perform the registration.
6. Select the type of request "Participant connection" for legal or natural person.
7. Register on the resource by filling in all the required fields of the form. If you want to mine, check the box **Please grant mining rights**.
8. Enter the transaction ID of the token lease transfer in the **Proof of WEST token ownership** field.
9. Please, wait for the connection application consideration. You can start working in the "Waves Enterprise Mainnet" after successful registration.
10. *Run* the node after obtaining permission and getting a license to connect to the network "Waves Enterprise Mainnet", public key of which you specified in the application.
11. Transfer or lease tokens to the address of the connected node for the mining and work in the network.

20.1.2 Fees in the "Waves Enterprise Mainnet"

#	Transaction type	Fee	Description
1	*Genesis transaction*	no fee	Initial binding of the balance to the addresses of nodes created at the start of the blockchain
2	Payment Transaction (not in use)		
3	*Issue Transaction*	1WEST	Tokens issue. Commission is taken only in WEST
4	*Transfer Transaction*	0.01WEST	Tokens transfer
5	*Reissue Transaction*	1WEST	Tokens reissue
6	*Burn Transaction*	0.05WEST	Tokens burn
8	*Lease Transaction*	0.01WEST	Tokens lease
9	*Lease Cancel Transaction*	0.01WEST	Cancel of the tokens lease
10	*Create Alias Transaction*	1WEST	Alias creation
11	*MassTransfer Transaction*	0.05WEST	Mass tokens transfer. Minimum commission is specified, a commission amount depends on a number of addresses in a transaction
12	*Data Transaction*	0.05WEST	Transaction with the data in the keyvalue pairs format. The fee is always charged |**br**| to the transaction author. Minimum commission is specified, the fee depends on data volume
13	*SetScript Transaction*	0.5WEST	Transaction which is binding a script with a RIDE contract to an account
14	SponsorFee Transaction (not in use)		
15	*SetAssetScript*	1WEST	Transaction which is binding a script with a RIDE contract to an asset
101	*Genesis Permission Transaction*	no fee	Assignment of the first network administrator for further distribution of rights
102	*Permission Transaction*	0.01WEST	Issuance/withdrawal of rights from the account
103	*CreateContract Transaction*	1WEST	Dockercontract creation
104	*CallContract Transaction*	0.1WEST	Dockercontract call
105	*ExecutedContract Transaction*	no fee	Dockercontract execution
106	*DisableContract Transaction*	0.01WEST	Dockercontract disable
107	*UpdateContract Transaction*	1WEST	Dockercontract update
110	*GenesisRegisterNode Transaction*	no fee	Node registration in the genesis block with the blockchain start
111	*RegisterNode Transaction*	0.01WEST	New node registration
112	*CreatePolicy*	1WEST	Access group creation

20.2 Working inside the "Waves Enterprise Partnernet"

20.2.1 Connection of the node to the " Waves Enterprise Partnernet"

Follow these steps for the node connection to the "Waves Enterprise Partnernet":

1. *Deploy* a single node as it is for the "Waves Enterprise Mainnet" connection.
2. Go to the Waves Enterprise support website and perform the registration.
3. Select the type of request "Participant connection" for legal or natural person.
4. Register on the resource by filling in all the required fields of the form. If you want to mine, check the box **Please grant mining rights**.
5. Please, wait for the connection application consideration. You can start working in the "Waves Enterprise Partnernet" after successful registration and getting a license.
6. *Run* the node after getting the application approve.

CHAPTER

TWENTYONE

API TOOLS OF THE NODE

The Waves Enterprise blockchain platform gives the opportunity of coordination with the blockchain for obtaining of data (transactions, blocks, balances, etc.), as well as for writing of information into the blockchain (signing and sending of transactions) with the use of the gRPC interface and the RESTful API of the node.

21.1 gRPC

gRPC is a highperformance remote procedure call (RPC) framework that runs on top of HTTP/2. It uses the Protobuf protocol as a data type description and serialization tool. Officially, the *gRPC <https://grpc.io/>* framework supports 10 programming languages. The list of the languages in use is available in the official gRPC documentation.

21.1.1 How to use the gRPC framework

Prior to operation of the gRPC interface, following preparatory measures have to be taken:

1. Decide on the programming language in which to interact with the node.
2. Install the fRPC framework.
3. Download the protobuf files containing the data structure for development of requests to the node or smart contracts from the GitHub page of the project.

On this page, the archive `weeventsproto.zip` is available, in which the used version of the node and corresponding files should be picked.

The protoc plugin of the gRPC framework is used for generation of code based on the data structure contained in the protobuf files.

The gRPC interface is enabled and configured via the *node configuration file*. In order to interact with the node, the **6865** port is used.

The gRPC interface of the platform is used for the following tasks:

- Tracking of events in the blockchain.
- Realization of signature methods with certificates (PKI).
- Realization of encryption methods.
- Obtaining of information about a transaction according to its ID.
- Obtaining of node configuration parameters.
- Obtaining of information about state of a smart contract

For each of this tasks separate sets of methods are used:

- gRPC methods of the node
- *gRPC servicesused by smart contracts.*

21.2 REST API

The REST API allows users to interact remotely with the node via JSON queries and responses. Interaction with the API is performed via the https protocol. The famous Swagger framework serves an the API interface.

21.2.1 Node REST API methods

Full description of the REST API methods you can find on the API Docs page. Almost all REST API methods are closed by the *authorization*. If a method is opened, you'll see the badge .

Activation

Hint: The rules for generating requests to the node are given in module *How to use the REST API*.

GET /activation/status

Returns the activation status of the new functionality in the node(s).

Method Response:


```
{"height": 47041,
 "votingInterval": 1,
 "votingThreshold": 1,
 "nextCheck": 47041,
 "features": [
    {"id": 1,
     "description": "Minimum Generating Balance of 1000 WEST",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 2,
     "description": "NG Protocol",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 3,
     "description": "Mass Transfer Transaction",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 4,
     "description": "Smart Accounts",
```


(continues on next page)

(continued from previous page)


```
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 },
   {"id": 5,
    "description": "Data Transaction",
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 },
   {"id": 6,
    "description": "Burn Any Tokens",
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 },
   {"id": 7,
    "description": "Fee Sponsorship",
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 },
   {"id": 8,
    "description": "Fair PoS",
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 },
   {"id": 9,
    "description": "Smart Assets",
    "blockchainStatus": "VOTING",
    "nodeStatus": "IMPLEMENTED",
    "supportingBlocks": 0 },
   {"id": 10,
    "description": "Smart Account Trading",
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 } ]
}
```


Addresses

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /addresses/info/{address}

Getting a public key by the address. The method returns only those public keys that are stored in the `keystore.dat` file of the node.

Method Response:


```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt"
}
```

GET/addresses

Get all addresses of participants whose key pairs are stored in the node keystore.

Method Response:

```
[
  "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```

GET/addresses/seq/{from}/{to}

Gets all addresses of participants whose key pairs are stored in node keystore in the specified range.

Method Response:

```
[
  "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```

GET/addresses/balance/{address}

Get the balance for the address {address}.

Method Response:

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "confirmations": 0,
  "balance": 100945889661986
}
```

POST/addresses/balance/details

Get balances for the address list.

Method Query:

```
{
   "addresses": [
   "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ", "3N11u447zghwj9MemYkrkt9v9xDaMwTY9nG"
   ]
}
```

GET/addresses/effectivebalance/{address}/{confirmations}

Get the balance for the address {address} after a number of confirmations > = value {confirmations}. Returns the total balance of the participant, including assets transferred to the participant for the leasing.

Method Response:

```
{
  "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "confirmations": 1,
  "balance": 0
}
```

GET /addresses/effectiveBalance/{address}

Get the effective balance of the specified address.

Method Response

```
{
   "address": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
   "confirmations": 0,
   "balance": 1240001592820000
}
```

GET /addresses/generatingBalance/{address}/at/{height}

Returns the generating balance of an address at the specified height.

Note: The method shows the generating balance determined at a height not lower than 2000 blocks ago.

Method Query:

```
{
  "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "height": 1000
}
```

Method Response:

```
{
  "address": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "generatingBalance": 1011543800600
}
```

GET/addresses/balance/details/{address}

Returns detailed information about balance of address {address}.

Method Query:

```
{
  "addresses": [
    "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ"
  ]
}
```

Method Response:

```
[
  {
    "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "regular": 0,
    "generating": 0,
    "available": 0,
    "effective": 0
  }
]
```

Response Options

- Regular total balance of participant, including assets transferred for leasing
- Available total balance of participant, except for assets transferred for leasing
- Effective — total balance of participant, including assets transferred to participant for leasing (Available + assets transferred to you for leasing)
- Generating minimum balance of participant, including assets transferred to participant for leasing, for the last 1000 blocks (used for mining)

GET/addresses/scriptInfo/{address}

Get information about the script installed on the address {address}.

Method Response:

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "script":
↪"3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
↪",
  "scriptText": "ScriptV1(BLOCK(LET(x,CONST_LONG(1)),FUNCTION_CALL(FunctionHeader(==,List(LONG,␣
↪LONG)),List(FUNCTION_CALL(FunctionHeader(+,List(LONG, LONG)),List(REF(x,LONG), CONST_LONG(1)),
↪LONG), CONST_LONG(2)),BOOLEAN),BOOLEAN))",
  "complexity": 11,
  "extraFee": 10001
}
```

Response Options

- “address” address in Base58 format
- “script” Base64 representation of the script

- "scriptText" source code of the script
- "complexity" complexity of the script
- "extraFee" fee for outgoing transactions set by the script

POST/addresses/sign/{address}

Returns the message encoded in BASE58 format signed by address private key {address}, stored in node keystore. The message is first signed and then converted.

Method Query:

```
{
 "message": "mytext"
}
```

Method Response:

```
{
  "message": "wWshKhJj",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"62PFG855ThsEHUZ4N8VE8kMyHCK9GWnvtTZ3hq6JHYv12BhP1eRjegA6nSa3DAoTTMammhamadvizDUYZAZtKY9S"
}
```

POST/addresses/verify/{address}

Validates signature of a message executed by address {address}, including the one created through POST method/addresses/sign/{address}.

Method Query:

```
{
  "message": "wWshKhJj",
  "publickey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts"
}
```

Method Response:

```
{
  "valid": true
}
```

POST/addresses/signtext/{address}

Returns a message signed by address private key {address} stored in the node keystore.

Method Query:

```
{
  "message": "mytext"
}
```

Method Response:

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7Zjnrzs9QYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u"
}
```

POST/addresses/verifytext/{address}

Validates signature of a message executed by address {address}, including the one created through the POSTmethod /addresses/signtext/{address}.

Method Query:

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7Zjnrzs9QYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u"
}
```

Method Response:

```
{
  "valid": true
}
```

GET /addresses/validate/{addressOrAlias}

Validates correctness of specified address or its alias {addressOrAlias} in a network blockchain of operating node.

Method Response:

```
{
  addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
  valid: true
}
```

POST /addresses/validateMany

Checks the validity of addresses or aliases.

Method Query:

```
{
  addressesOrAliases: [
    "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
    "alias:T:asdfghjk",
    "alias:T:1nvAliDAl1ass99911%^&$$$ "
  ]
}
```

Method Response:

```
{
  validations: [
    {
      addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
      valid: true
    },
    {
      addressOrAlias: "alias:T:asdfghjk",
      valid: true
    },
    {
      addressOrAlias: "alias:T:1nvAliDAl1ass99911%^&$$$ ",
      valid: false,
      reason: "GenericError(Alias should contain only following characters: -.0123456789@_
↪abcdefghijklmnopqrstuvwxyz)"
    }
  ]
}
```

GET /addresses/publicKey/{publicKey}

Returns participant address based on its public key.

Method Response:

```
{
  "address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq"
}
```

GET/addresses/data/{address}

Returns all data recorded to address account {address}.

Method Response:

```
[
  {
    "key": "4yR7b6Gv2rzLrhYBHpgVCmLH42raPGTF4Ggi1N36aWnY",
    "type": "integer",
```

(continues on next page)

(continued from previous page)

```
    "value": 1500000
  }
]
```

GET/addresses/data/{address}/{key}

Returns data recorded to address account {address} by key {key}.

Method Response:

```
{
  "key": "4yR7b6Gv2rzLrhYBHpgVCmLH42raPGTF4Ggi1N36aWnY",
  "type": "integer",
  "value": 1500000
}
```

Alias

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET/alias/byalias/{alias}

Gets participant address by its alias {alias}.

Method Response:

```
{

     "address": "address:3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"

}
```

GET /alias/byaddress/{address}

Gets alias {alias} of participant by its address {address}.

Method Response:

```
[
    "alias:HUMANREADABLE1",
    "alias:HUMANREADABLE2",
    "alias:HUMANREADABLE3",
]
```

Anchoring

GET /anchoring/config

Hint: Rules of the creating requests to a node, see *How to use the REST API* section.

Get the *anchoring* section of the node configuration file.

Method answer


```
 {
"enabled": true,
"currentChainOwnerAddress": "3FWwx4o1177A4oeHAEW5EQ6Bkn4Lv48quYz",
"mainnetNodeAddress": "https://clinton-pool.wavesenterpriseservices.com:443",
"mainnetSchemeByte": "L",
"mainnetRecipientAddress": "3JzVWCSV6v4ucSxtGSjZsvdiCT1FAzwpqrP",
"mainnetFee": 8000000,
"currentChainFee": 666666,
"heightRange": 5,
"heightAbove": 3,
"threshold": 10
 }
```


Assets

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET/assets/balance/{address}

Returns balance of all address {address} assets.

Method Response:


```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
      "balance": 4879179221,
      "quantity": 48791792210,
      "reissuable": true,
      "minSponsoredAssetFee" : 100,
      "sponsorBalance" : 1233221,
      "issueTransaction" : {
         "type" : 3,
         ...
```


(continues on next page)

(continued from previous page)

```
      }
    },
    {
      "assetId": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
      "balance": 10,
      "quantity": 10000000000,
      "reissuable": false,
      "issueTransaction" : {
         "type" : 3,
         ...
      }
    }
  ]
}
```

Method Parameters:

- "Address" participant address
- "balances" object with participant balance
- "assetId" asset ID
- "balance" asset balance
- "quantity" number of issued assets
- "reissuable" indicator whether asset can be reissued or not
- "issueTransaction" asset creation transaction
- "minSponsoredAssetFee" minimum value of fee for sponsorship transactions
- "sponsorBalance" assets allocated for payment of sponsored asset transactions

GET /assets/balance/{address}/{assetId}

Returns address {address} balance by asset {assetId}.

Method Response:

```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
  "balance": 4879179221
}
```

GET /assets/details/{assetId}

Returns description of asset {assetId}.

Method Response:

```
{
  "assetId" : "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ",
  "issueHeight" : 140194,
  "issueTimestamp" : 1504015013373,
```

(continues on next page)

(continued from previous page)

```
  "issuer" : "3NCBMxgdghg4tUhEEffSXy11L6hUi6fcBpd",
  "name" : "name",
  "description" : "Sponsored asset",
  "decimals" : 1,
  "reissuable" : true,
  "quantity" : 1221905614,
  "script" : null,
  "scriptText" : null,
  "complexity" : 0,
  "extraFee": 0,
  "minSponsoredAssetFee" : 100000 // null assume no sponsorship, number - amount of assets for␣
↪minimal fee
}
```

GET /assets/{assetId}/distribution

Returns distribution of asset {assetId}.

Method Response:

```
{
  "3P8GxcTEyZtG6LEfnn9knp9wu8uLKrAFHCb": 1,
  "3P2voHxcJg79csj4YspNq1akepX8TSmGhTE": 1200
}
```

POST /assets/balance

Returns the assets balance for one or few addresses.

Method Response

```
{
  "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG": [],
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn": []
}
```

Blocks

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

The last block may contain a different number of transactions during the period of its creation. It depends on the fact that while the block is not accepted by the nodesminers, the number of transactions in it can constantly change. Therefore, when using methods that provide information about the last block, it should be kept in mind that the number of transactions in the last block may change.

GET /blocks/height

Returns block number of current blockchain state.

Method Response:

```
{
  "height": 7788
}
```

GET /blocks/height/{signature}

Returns height (number) of block by its signature.

GET /blocks/first

Returns contents of first block (genesis block).

GET /blocks/last

Returns contents of last block.

Method Response:

```
{
  "version": 2,
  "timestamp": 1479313809528,
  "reference":
↪"4MLXQDbARiJDEAoy5vZ8QYh1yNnDhdGhGWkDKna8J6QXb7agVpFEi16hHBGUxxnq8x4myG4w66DR4Ze8FM5dh8Gi",
  "nxtconsensus": {
    "basetarget": 464,
    "generationsignature": "7WUV2TufaRAyjiCPFdnAWbn2Q7Jk7nBmWbnnDXKDEeJv"
  },
  "transactions": [
    {
      "type": 2,
      "id":
↪"64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "fee": 100000,
      "timestamp": 1479313757194,
      "signature":
↪"64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
      "recipient": "3N8UPtqiy322NVr1fLP7SaK1AaCU7oPaVuy",
      "amount": 1000000000
    }
  ],
  "generator": "3N5GRqzDBhjVXnCn44baHcz2GoZy5qLxtTh",
  "signature":
↪"4ZhZdLAvaGneLU4K4b2eTgRQvbBjEZrtwo1qAhM9ar3A3weGEutbfNKM4WJ9JZnV8BXenx8JRGVNwpfxf3prGaxd",
  "fee": 100000,
  "blocksize": 369
}
```

GET /blocks/at/{height}

Returns contents of block at height {height}.

GET /blocks/seq/{from}/{to}

Returns contents of blocks ranging from {from} to {to}.

GET /blocks/seqext/{from}/{to}

Returns contents of blocks with additional transactions info ranging from {from} to {to}.

GET /blocks/signature/{signature}

Returns contents of block by its signature {signature}.

GET /blocks/address/{address}/{from}/{to}

Returns all blocks generated (mined) by address {address}.

GET /blocks/child/{signature}

Returns block inherited from block with signature {signature}.

GET /blocks/headers/at/{height}

Returns block header at height {height}.

GET /blocks/headers/seq/{from}/{to}

Returns block headers ranging from {from} to {to}.

GET /blocks/headers/last

Returns header of last block in the blockchain.

Consensus

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /consensus/algo

Returns type of consensus algorithm used on the network.

Method Response:

```
{
    "consensusAlgo": "Fair Proof-of-Stake (FairPoS)"
}
```

GET /consensus/settings

Returns consensus settings specified in node configuration file.

Method Response:

```
{
    "consensusAlgo": "Proof-of-Authority (PoA)",
    "roundDuration": "25 seconds",
    "syncDuration": "5 seconds",
    "banDurationBlocks": 50,
    "warningsForBan": 3
}
```

GET /consensus/minersAtHeight/{height}

Returns miner queue at height {height}.

Method Response:

```
{
"miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
],
"height": 1
}
```

GET /consensus/miners/{timestamp}

Returns miner queue at timestamp {timestamp}.

Method Response:


```
{
    "miners": [
        "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
        "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
        "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
        "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
        "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
    ],
    "timestamp": 1547804621000
}
```


GET /consensus/bannedMiners/{height}

Returns a list of blocked miners at height {height}.

Method Response:


```
{
    "bannedMiners": [],
    "height": 1000
}
```


GET /consensus/basetarget/{blockId}

Returns value of 'base complexity' _ (basetarget) of creating block {blockId} .

GET /consensus/basetarget

Returns value of 'base complexity'_ (basetarget) of creating last block.

GET /consensus/generatingbalance/{address}

Returns generating balance available for minning node {address} minimum participant balance including assets transferred to participant for leasing, for last 1000 blocks.

GET /consensus/generationsignature/{blockId}

Returns value of 'generation signature'_ of generating block {blockId}.

GET /consensus/generationsignature

Returns value of 'generation signature'_ of last block.

Contracts

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /contracts

Returns the contracts info.

Method Response

```
[
  {
    "contractId": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "image": "registry.wvservices.com/wv-sc/may14_1:latest",
    "imageHash": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957",
    "version": 1,
    "active": false
   }
 ]
```

POST /contracts

Returns some parameters for the one or more contract IDs specified in the query.

Request parameters:

```
{
"contracts": [
  "string"
  ]
}
```

Method Response

```
{
  "8vBJhy4eS8oEwCHC3yS3M6nZd5CLBa6XNt4Nk3yEEExG": [
   {
   "type": "string",
   "value": "Only description",
   "key": "Description"
```

(continues on next page)

(continued from previous page)

```
  },
  {
  "type": "integer",
  "value": -9223372036854776000,
  "key": "key_may"
  }
 ]
}
```

GET /contracts/info/{contractId}

Returns current information about specified contract version, contract location, and the image hash.

Request parameters:

```
"Contract id"
```

Method Response

```
[
  {
    "contractId": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "image": "registry.wvservices.com/wv-sc/may14_1:latest",
    "imageHash": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957",
    "version": 1,
    "active": false
  }
]
```

GET /contracts/status/{id}

Returns the contract execution transaction status.

Request parameters:

```
"id" - Transaction ID
```

Method Response

```
[
  {
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
    "status": "Success",
    "code": null,
    "message": "Smart contract transaction successfully mined",
    "timestamp": 1558961372834,
    "signature":
↪"4gXy7qtzkaHHH6NkksnZ5pnv8juF65MvjQ9JgVztpgNwLNwuyyr27Db3gCh5YyADqZeBH72EyAkBouUoKvwJ3RQJ"
  },
  }
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
```

(continues on next page)

(continued from previous page)

```
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
    "status": "Success",
    "code": null,
    "message": "Smart contract transaction successfully mined",
    "timestamp": 1558961376012,
    "signature":
↪"3Vhqc9DvNhMvFFtWnBuV4XwQ62ZcTAvLNZYmeGc7mGzMcnGZ3RLshDs393fnQu1WTh8CmL58YnvnjyULEEi5yorV"
  }
]
```

GET /contracts/{contractId}

Returns result of smart contract execution by its ID (contract creation transaction ID).

Request parameters

```
"contractId" - Contract ID
"offset" - Offset number
"matches" - String for matches search
"limit" - Limit number
```

Method Response:

```
[
  {
    "key": "avg",
    "type": "string",
    "value": "3897.80146957"
  },
  {
    "key": "buy_price",
    "type": "string",
    "value": "3842"
  }
]
```

POST /contracts/{contractId}

Returns keys of smart contracts by its ID (contract creation transaction ID).

Request parameters

```
 "Contract Id"
 {
"keys": [
  "string"
 ]
}
```

Method Response:

```
[
  {
    "type": "string",
    "key": "avg",
    "value": "3897.80146957"
  },
  {
    "type": "string",
    "key": "buy_price",
    "value": "3842"
  }
]
```


GET /contracts/executedtxfor/{id}

Returns result of smart contract execution by ID of contract execution transaction.

Request parameters:

```
"id" - Transaction ID
```

Method Response:


```
{
  "type": 105,
  "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    "4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "version": 1,
  "tx": {
      "type": 103,
      "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
      "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
      "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
      "fee": 500000,
      "timestamp": 1550591678479,
      "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
      "version": 1,
      "image": "stateful-increment-contract:latest",
      "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
      "contractName": "stateful-increment-contract",
      "params": [],
      "height": 1619
  },
  "results": []
}
```

GET /contracts/{contractId}/{key}

Returns smart contract execution value by its ID (contract creation transaction ID) and key {key}.

Request parameters:

```
"Contract id"
"key" - Key name
```

Method Response:

```
{
  "key": "updated",
  "type": "integer",
  "value": 1545835909
}
```

Crypto

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

POST /crypto/encryptSeparate

Encrypts the text separately for the each recipient with the unique key.

Method Query

```
{
 "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
 "password": "some string as a password",
 "encryptionText": "some text to encrypt",
 "recipientsPublicKeys": [
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
 "9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"]
}
```

Method Response

```
{
 "encryptedText": "IZ5Kk5YNspMWl/jmlTizVxD6Nik=",
 "publicKey":
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
 "wrappedKey":
↪"uWVoxJAzruwTDDSbphDS31TjSQX6CSWXivp3x34uE3XtnMqqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfBbwR"
},
{
 "encryptedText": "F9uO1ORGvSEDe6dWm1pzJQ+3xqE=",
 "publicKey":
↪"9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S",
 "wrappedKey":
↪"LdzdoKadUzBTMwczGYgu1AM4YrbbLr9Uh1MvQ3MPcLZUhCD9herz4dv1m6ssaVHPiBNUGgqKnLZ6Si4Cc64UvhXBbG"
}
```

POST /crypto/encryptCommon

Encrypts the data with a single CEK key for all recipients and the CEK wraps into a unique KEK for the each recipient.

Method Query

```
{
 "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
 "password": "some string as a password",
 "encryptionText": "some text to encrypt",
 "recipientsPublicKeys": [
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc",
 "9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"]
}
```

Method Response

```
{
"encryptedText": "NpCCig2i3jzo0xBnfqjfedbti8Y=",
"recipientToWrappedStructure": {
  "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc":
"M8pAe8HnKiWLE1HsC1ML5t8b7giWxiHfvagh7Y3F7rZL8q1tqMCJMYJo4qz4b3xjcuuUiV57tY3k7oSig53Aw1Dkkw",
  "9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S":
"Doqn6gPvBBeSu2vdwgFYMbDHM4knEGMbqPn8Np76mNRRoZXLDioofyVbSSaTTEr4cvXwzEwVMugiy2wuzFWk3zCiT3"
 }
}
```

POST /crypto/decrypt

Decrypts the data. The decryption is available only if the message recipient's key is in the node's keystore.

Method Query

```
{
 "recipient": "3M5F8B1qxSY1W6kA2ZnQiDB4JTGz9W1jvQy",
 "password": "some string as a password",
 "encryptedText": "oiKFJijfid8HkjsjdhKHhud987d",
 "wrappedKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy"
 "senderPublicKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy",
}
```

Method Response

```
{
"decryptedText": "some string for encryption",
}
```

Debug

Hint: The rules for generating node queries are given in module *How to use the REST API*.

GET /debug/blocks/{howMany}

Gets sizes and full hashes for last blocks. The blocks number is specified during the request.

Method Response


```
[
    {
        "226": "7CkZxrAjU8bnat8CjVAPagobNYazyv1HASubmp7YYqGe"
    },
    {
        "226": "GS3y9fUHAKCamq52TPsjizDVir8J7iGoe8P2XZLasxsC"
    },
    {
        "226": "B9LmhGGDdvcfUA9JEWvyVrT9sazZE6gibpAN13xUN7KV"
    },
    {
        "226": "Byb9MHtwYf3MFyi2tbhQ3GTdCct5phKq9REkbjQTzdne"
    },
    {
        "226": "HSxSHbiV4tZc8RaN6jxdhgtkAhjxuLn76uHxerMRUefA"
    }
]
```


GET /debug/info

Shows all information for the debugging and testing.

Method Response


```
{
    "stateHeight": 74015,
    "extensionLoaderState": "State(Idle)",
    "historyReplierCacheSizes": {
        "blocks": 13,
        "microBlocks": 2
    },
    "microBlockSynchronizerCacheSizes": {
        "microBlockOwners": 0,
        "nextInventories": 0,
        "awaiting": 0,
        "successfullyReceived": 0
    },
    "scoreObserverStats": {
        "localScore": 42142328633037120000,
        "scoresCacheSize": 4
    },
```


(continues on next page)

(continued from previous page)

```
    "minerState": "mining microblocks"
}
```

POST /debug/rollback

Removes all blocks after given height.

Sample response

```
{
    "rollbackTo": 100,
    "returnTransactionsToUtx": true
}
```

Method Response

```
{
    "BlockId":
↪"4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv"
}
```

POST /debug/validate

Validates a transaction and measures time spent in milliseconds.

Query Parameters

```
"id" - Transaction ID
```

Method Response

```
{
    "valid": false,
    "validationTime": 14444
}
```

GET /debug/minerInfo

Shows all miner information for debugging.

Method Response

```
[
    {
        "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
        "miningBalance": 1248959867200000,
        "timestamp": 1585923248329
    }
]
```

GET /debug/historyInfo

Shows all last block history for debugging.

Method Response


```
{
"lastBlockIds": [
"37P4fvexYHPUzNPRRqYbRYxGz7x3r5jFznck7amaS6aWnHL5oQqrqCzsSh1HvYKnd2ZhU6n6sWYPb3hxsY8FBfmZ",
"5RRu1qtesz4KvrVp4fxzQHebq2fRanNsg3HJKwD4uChqySm7vFHCdHKU6iZYXJDVmfSxiE9Maeb6sM2JireaWLbx",
"3Lo27JfjekcZnJsYEe7st7evDZ6TgmCUBtiZrSxUCobKL48DZQ4dXMfp89WYjEykH15HEHSXzqMSTQigE8vEcN2r",
"r4RuxEXAqgfDMKVXRWmZcGMaWKDsAvVxfXDtw8d6bamLR61J1gaoesargYSoZQqRbDrBcefLprk7D78fA728719",
"3F4Up46crZbpKVWUeieL6GeSrVMYm7JJ7aX6aHD6B8wedFggSKv8d3H39Qy9MLEauFBU9m3qZV1U8emhmnqwmLbg",
"QSuBkEtVe9nik5T5S33ogeCbgKy7ihBkS2pwYayK23m4ANier83ThpajEzvpbyPy9pPWZc5St8mYUKxXDscKuRC",
"4udpNnz3e1M1GbVZxtwfg8gpF6EbiKxRCRBwi6iRMyLsvh5J2Ec9Wqyu2sq2KYL75o12yiP8TszworeUfuxNmJ5g",
"5BZYZ4RZAJjM5KKCaHpyUsXnb4uunnM5kcfTojc5QzQo3vyP2w3YD4qrALizkkQQR4ziS77BoAGb56QCecUtHFFN",
"5JwfLaF1oGxRXVCdDbFuKpxrvxgLCGU3kCFwxUhLL8G3xV211MrKBuAuQ4MaC5uN574uV9U8M6HfHTMERnfr5jGJ",
"4bysMhz14E1rC7dLYScfVVqPmHqzi8jdhcnkruJmCNL86TwV2cbF7G9YVchvTrv9qbQZ7JQownV59gRRcD26zm16"
],
"microBlockIds": []
}
```


GET /debug/configInfo

Shows currently running node config.

Method Response


```
  {
"node": {
  "anchoring": {
    "enable": "no"
  },
  "blockchain": {
    "consensus": {
      "type": "pos"
    },
    "custom": {
      "address-scheme-character": "K",
      "functionality": {
        "blocks-for-feature-activation": 10,
        "feature-check-blocks-period": 30,
        "pre-activated-features": {  ...
  ......................................
  "wallet": {
    "file": "wallet.dat",
    "password": ""
  },
  "waves-crypto": "yes"
  }
  }
```

DELETE /debug/rollbackto/{signature}

Rollbacks the state to the block with a given signature.

Query Parameters

```
"signature" - Block signature
```

Method Response

```
{
    "BlockId":
↪"4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv"
}
```

GET /debug/portfolios/{address}

Gets current portfolio considering pessimistic transactions in the UTX pool.

Query Parameters

```
"address" - Node address
```

Method Response

```
{
    "balance": 104665861710336,
    "lease": {
        "in": 0,
        "out": 0
    },
    "assets": {}
}
```

POST /debug/print

Prints a string at DEBUG level, strips to 250 chars.

Sample response

```
{
    "message": "string"
}
```

GET /debug/state

Gets current state of the node.

Method Response


```
{
    "3JD3qDmgL1icDaxa3n24YSjxr9Jze5MBVVs": 4899000000,
    "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
    "3JU5tCoswHH7FKPBUowySWBnQwpbZiYyNhB": 300021381800000,
    "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000362600000,
    "3JEW9XnPC8w3qQ4AJyVTDBmsVUp32QKoCGD": 5000000000,
    "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
    "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248938560600000,
    "3JV6V4JEVc3a9uSqRmdUMvMKMfZa16HbGmq": 4770000000,
    "3JZtYeGEZHjb2zQ6EcSEo524PdafPn6vWkc": 900000000,
    "3JMMFLX9d1rmXaBK9AF7Wuwzu4vRkkoVQBC": 4670000000,
    "3JJDpPDqSPokKp5jEmzwMzmaPUyopLZjW1C": 800000000,
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 994280900000
}
```


GET /debug/stateWE/{height}

Gets state at specified height.

Query Parameters

```
"height" - Block height
```

Method Response


```
{
    "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
    "3JU5tCoswHH7FKPBUowySWBnQwpbZiYyNhB": 300020907600000,
    "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000350600000,
    "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
    "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248960085800000,
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 994280900000
}
```


Leasing

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /leasing/active/{address}

Returns list of lease creation transactions, in which {address} was involved as sender or recipient.

Method Response:


```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEWoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLYeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
↪"25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPFyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```



```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEWoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLYeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
↪"25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPFyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```


Licenses

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /licenses

Returns a list of all downloaded licenses.

Method Response:


```
[
  {
    "license": {
      "version": 1,
      "id": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
      "license_type": null,
      "issued_at": "2020-02-27T16:11:14.784Z",
      "node_owner_address": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
      "valid_from": "2020-02-20",
      "valid_to": "2020-02-27",
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "signature":
↪"ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957mLT1ippM7tmfSC8u",
    "signer_id": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957"
  },
  {
    "license": {
      "version": 1,
      "id": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
      "license_type": null,
      "issued_at": "2020-02-27T16:12:34.327Z",
      "node_owner_address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq",
      "valid_from": "2020-02-29",
      "valid_to": null,
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "signature":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts",
    "signer_id": "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ"
  }
]
```


GET /licenses/status

Returns the node license activation status

Method Response:


```
{
  "status" : "TRIAL",
  "description" : "Trial period is active. Blocks before expiration: '{num}'"
}
```

POST /licenses/upload

Adds a new license in JSON format in the node

Method request

```
{
  "license": {
    "version": 1,
    "id": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
    "license_type": null,
    "issued_at": "2020-02-27T16:12:34.327Z",
    "node_owner_address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq",
    "valid_from": "2020-02-29",
    "valid_to": null,
    "features": [
      "all_inclusive"
    ]
  },
  "signer_public_key": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts",
  "signer_id": "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ"
}
```

Method Response:

```
{
    "message": "License upload successfully"
}
```

Node

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /node/config

Returns main node configuration parameters.

Method Response:

```
{
    {
      "version": "1.3.0-RC7",
      "gostCrypto": false,
      "chainId": "V",
      "consensus": "POA",
      "minimumFee": {
          "3": 0,
          "4": 0,
          "5": 0,
```

(continues on next page)

(continued from previous page)


```
            "6": 0,
            "7": 0,
            "8": 0,
            "9": 0,
            "10": 0,
            "11": 0,
            "12": 0,
            "13": 0,
            "14": 0,
            "15": 0,
            "102": 0,
            "103": 0,
            "104": 0,
            "106": 0,
            "107": 0,
            "111": 0,
            "112": 0,
            "113": 0,
            "114": 0
        },
        "additionalFee": {
            "11": 0,
            "12": 0
        },
        "maxTransactionsInMicroBlock": 500,
        "minMicroBlockAge": 0,
        "microBlockInterval": 1000,
        "blockTiming": {
            "roundDuration": 7000,
            "syncDuration": 700
    }
}
```


GET /node/logging

Displays a list of loggers and their logging level for each one separately.

Method Response:

```
ROOT-DEBUG
akka-DEBUG
akka.actor-DEBUG
akka.actor.ActorSystemImpl-DEBUG
akka.event-DEBUG
akka.event.slf4j-DEBUG
akka.event.slf4j.Slf4jLogger-DEBUG
com-DEBUG
com.github-DEBUG
com.github.dockerjava-DEBUG
com.github.dockerjava.core-DEBUG
com.github.dockerjava.core.command-DEBUG
com.github.dockerjava.core.command.AbstrDockerCmd-DEBUG
com.github.dockerjava.core.exec-DEBUG
```

POST /node/logging

Sets a specific logging level for the defined loggers.

Method Request:

```
{
    "logger": "com.wavesplatform.Application",
    "level": "ALL"
}
```

Method Response:

POST /node/stop

Query stops node.

GET /node/status

Returns main node configuration parameters.

Method Response:

```
{
    "blockchainHeight": 47041,
    "stateHeight": 47041,
    "updatedTimestamp": 1544709501138,
    "updatedDate": "2018-12-13T13:58:21.138Z"
}
```

Note: In case of any errors during usage the GOST cryptography on the node, this method will indicate possible errors with JCP:

```
{
    "error": 199,
    "message": "Environment check failed: Supported JCSP version is 5.0.40424, actual is 2.0.40424"
}
```

GET /node/version

Returns version of application.

Method Response:

```
{
  "version": "Waves Enterprise v0.9.0"
}
```

GET /node/owner

Returns the address and public key of the node owner.

Method Response:

```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt"
}
```

Peers

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

POST /peers/connect

Request to connect a new host to the node.

Method Query:

```
{
    "host":"127.0.0.1",
    "port":"9084"
}
```

Method Response:

```
{
  "hostname": "localhost",
  "status": "Trying to connect"
}
```

GET /peers/connected

Returns a list of connected nodes.

Method Response:

```
{
  "peers": [
    {
      "address": "52.51.92.182/52.51.92.182:6863",
      "declaredAddress": "N/A",
      "peerName": "zx 182",
      "peerNonce": 183759
    },
    {
      "address": "ec2-52-28-66-217.eu-central-1.compute.amazonaws.com/52.28.66.217:6863",
      "declaredAddress": "N/A",
```

(continues on next page)

(continued from previous page)

```
      "peerName": "zx 217",
      "peerNonce": 1021800
    }
  ]
}
```

GET /peers/all

Returns a list of all known nodes.

Method Response:

```
{
  "peers": [
    {
      "address": "/13.80.103.153:6864",
      "lastSeen": 1544704874714
    }
  ]
}
```

GET /peers/suspended

Returns a list of suspended nodes.

Method Response:

```
[
  {
    "hostname": "/13.80.103.153",
    "timestamp": 1544704754619
  }
]
```

POST /peers/identity

Gets the public key of the peer which is used by the node for the connection and the confidential data transfer.

Method Query:

```
{
  "address": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "signature":
↪"6RwMUQcwrxtKDgM4ANes9Amu5EJgyfF9Bo6nTpXyD89ZKMAcpCM97igbWf2MmLXLdqNxdsUc68fd5TyRBEB6nqf"
}
```

Parameters:

- address the blockchain address corresponding to the “privacy.owneraddress” parameter in the node configuration file;
- signature electronic signature of the “address” field value.

Method Response:


```
{
  "publicKey": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8"
}
```


Parameters:

- publicKey the peer public key associated with "privacy.owneraddress" parameter in the configuration file. This parameter does not appear if the mode of the handshake checking turned off.

GET /peers/hostname/{address}

Gets the hostname and IP Address of the node by its address in the Waves Enterprise net.

Method Response:


```
{
  "hostname": "node1.we.io",
  "ip": "10.0.0.1"
}
```


GET /peers/allowedNodes

Gets the actual list of allowed participants at the request moment.

Method Response:


```
{
  "allowedNodes": [
    {
      "address": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
      "publicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY"
    },
    {
      "address": "3JLp8wt7rEUdn4Cca5Hp9jZ7w8T5XDAKicd",
      "publicKey": "J3ffCciVu3sustgb5vxmEHczACMR89Vty5ZBLbPn9xyg"
    },
    {
      "address": "3JRY1cp7atRMBd8QQoswRpH7DLawM5Pnk3L",
      "publicKey": "5vn4UcB9En1XgY6w2N6e9W7bqFshG4SL2RLFqEWEbWxG"
    }
  ],
  "timestamp": 1558697649489
}
```

Permissions

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

GET /permissions/{address}

Returns roles (permissions) assigned to specified address {address} which are valid at the moment.

Method Response:


```
{
    "roles": [
        {
        "role": "miner"
        },
        {
        "role": "permissioner"
        }
    ],
    "timestamp": 1544703449430
}
```


GET /permissions/{address}/at/{timestamp}

Returns roles (permissions) assigned to specified address {address} which are valid at the moment {timestamp}.

Method Response:


```
{
    "roles": [
        {
        "role": "miner"
        },
        {
        "role": "permissioner"
        }
    ],
    "timestamp": 1544703449430
}
```

POST /permissions/addresses

Returns roles (permissions) assigned to specified address list which are valid at the moment.

Method Query:

```
{
    "addresses": [
        "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w", "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7"
    ],
    "timestamp": 1544703449430
}
```

Method Response:

```
{
    "addressToRoles": [
        {
        "address": "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
        "roles": [
            {
            "role": "miner"
            },
            {
            "role": "permissioner"
            }
        ]
        },
        {
        "address": "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
        "roles": [
            {
            "role": "miner"
            }
        ]
        }
    ],
    "timestamp": 1544703449430
}
```

PKI

Warning: The PKI methods can be used only with GOST cryptography.

Digital signature formats listed in the table below is used in PKI. The digital signature number in the table is consistent for the `sigtype` field value.

Table 1: Digital signature formats

#	Digital signature format
1	CAdESBES
2	CAdESX Long Type 1
3	CAdEST

POST /pki/sign

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

This method creates a detached digital signature. `inputData` is data for generating a digital signature as an array of bytes in the **Base64** coding, `keystoreAlias` is a name of the key container of the digital signature private key. Also you need to specify a password in the `password` string.

Request example

```
{
"inputData" : "SGVsbG8gd29ybGQh",
"keystoreAlias" : "key1",
"password" : "password",
"sigType" : "CAdES_X_Long_Type_1",
}
```

Answer example

```
{
  "signature" :
↪"c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQjsndjfvnksdnjfn="
 }
```

GET /pki/keystoreAliases

This method returns all the keystore aliases based on the GOST cryptography.

Answer example

```
{
 [
  "3Mq9crNkTFf8oRPyisgtf4TjBvZxo4BL2ax",
  "e19a135e-11f7-4f0c-9109-a3d1c09812e3"
 ]
 }
```

POST /pki/verify

This method checks the detached digital signature for the sent data. The `extendedKeyUsageList` is optional and may contain an array of object identifiers OID. It is useful for the determination of the scope of the certificate. Any node with query parameters can check the certificate.

Request example

```
{
 "inputData" : "SGVsbG8gd29ybGQh",
 "signature" : "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQ=",
 "sigType" : "CAdES_X_Long_Type_1",
 "extendedKeyUsageList": [
 "1.2.643.7.1.1.1.1",
 "1.2.643.2.2.35.2"
```

(continues on next page)

(continued from previous page)

```
]
}
```

Answer example

```
{
 "sigStatus" : "true"
 }
```

Working with POST /pki/verify method

Using API *Post /pki/verify* method you can verify qualified digital signature. You need to install the root certificate on the node for proper using of API *Post /pki/verify*. The CA root certificate uniquely identifies the certification authority and is the basis in the chain of trust.

How to install a root certificate on a node

The root certificate is installing into the following Java directory:

```
-keystore /Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/jre/lib/
↪security/cacerts
```

The default password for the Java cacerts certificate store is `changeit`. You can change the password if you wish. Install certificates using the following command:

```
sudo keytool -import -alias testAliasCA_cryptopro -keystore /Library/Java/
↪JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/jre/lib/security/cacerts -file ~/
↪Downloads/cert.cer
```

Privacy

Hint: Rules of the creating requests to a node, see *How to use the REST API* section.

POST /privacy/sendData

Writing the confidential data to the node store.

Method request:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "info": {
    "filename":"Service contract #100/5.doc",
```

(continues on next page)

(continued from previous page)

```
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "some comments"
   },
   "data":
↪"TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhl
↪",
   "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
}
```

Parameters:

- sender blockchain address for data broadcast (corresponds the "privacy.owneraddress" parameter value in the node configuration file);
- password access password to the private key of the node keystore;
- policyId the group ID managing data forwarding;
- type the type of the data;
- info the information about the data;
- data строка, содержащая данные в формате **base64**;
- hash sha256хеш данных в формате **base58**.

Method answer:

```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
  "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc"
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
 }
```

POST /privacy/sendDataV2

The second version of the *POST /privacy/sendData* method allows to stream files with confidential data to the node storage.

Method request:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "hash": "e67ad392ab4d933f39d5234asdd96c18c491140e119d590103e7fd6de15623f9",
  "info": {
    "filename": "Договор об оказании услуг №100/5.doc",
```

(continues on next page)

(continued from previous page)


```
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "la la fam"
  },
  "fee": 15000000,
  "password": "12345qwert",
  "timestamp": 0
}
```


The parameters differ from the parameters of the *POST /privacy/sendData* method only in the absence of the `Data` field. You need to select and attach the data file in the corresponding Swagger window instead of filling the `Data` field.

Method answer:


```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
  "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc"
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
 }
```


GET /privacy/{policyid}/recipients

Getting all addresses of participants, signed to the access group {policyid}.

Method answer:


```
[
"3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
"3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```


GET /privacy/{policyid}/getHashes

Getting all addresses of participants, signed to the access group {policyid}.

Method answer:


```
[
  "3GCFaCWtvLDnC9yX29YftMbn75gwfdwGsBn",
  "3GGxcmNyq8ZAHzK7or14Ma84khwW8peBohJ",
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn",
  "3GKpShRQRTddF1yYhQ58ZnKMTnp2xdEzKqW"
]
```

GET /privacy/{policyid}/getHashes

Getting the array of identified hashes which are written with association to the {policyid}.

Method answer:

```
[
"FdfdNBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
"eedfdNBVqYXrapgJP9atQccdBPAgJPwHDKkh6A"
]
```

GET /privacy/{policyId}/getData/{policyItemHash}

Getting the confidential data package by its identified hash.

Method answer:

```
c29tZV9iYXNlNjRfZW5jb2RlZF9zdHJpbmc=
```

GET /privacy/{policyId}/getInfo/{policyItemHash}

Getting the metadata for the confidential data package by the identified hash.

Method answer:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policy": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "info": {
    "filename": "Contract №100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "Comment"
  },
"hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1"
}
```

POST /privacy/forceSync

Forced getting the confidential data package by the identified hash.

Method answer:

```
{
  "result": "success" // or "error"
  "message": "Address '3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8' not in policy 'policyName'"
}
```

POST /privacy/getInfos

Getting the meta information array about private data according with the provided group ID and data hash.

Request example:

```
{ "policiesDataHashes":
  [
   {
    "policyId": "somepolicyId_1",
    "datahashes": [ "datahash_1","datahash_2" ]
   },
   {
     "policyId": "somepolicyId_2",
     "datahashes": [ "datahash_3","datahash_4" ]
   }
  ]
 }
```

Method answer:

```
{
 "policiesDataInfo":[
    {
       "policyId":"somepolicyId_1",
       "datasInfo":[
          {
             "hash":"e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1",
             "sender":"3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
             "type":"file",
             "info":{
                "filename":"Contract №100/5.doc",
                "size":2048,
                "timestamp":1000000000,
                "author":"AIvanov@org.com",
                "comment":"Comment"
             }
          },
          {
             "hash":"e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1",
             "sender":"3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
             "type":"file",
             "info":{
                "filename":"Contract №101/5.doc",
                "size":"2048",
                "timestamp":1000000000,
                "author":"AIvanov@org.com",
                "comment":"Comment"
             }
          }
          }
        ]
    ]
}
```

Transactions

Hint: The rules for generating node queries are given in module *How to use the REST API*.

GET /transactions/info/{id}

Query transaction information by its ID.

Query Parameters:

```
"id" - Transaction ID
```

Method Response:

```
{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1549365736923,
  "attachment": "string",
  "signature":
↪"GkniccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm",
  "height": 7782
}
```

GET /transactions/address/{address}/limit/{limit}

Returns latest {limit} transactions from address {address}.

Method Response:

```
[
  [
    {
      "type": 2,
      "id":
↪"4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CdddnyBW",
      "fee": 100000,
      "timestamp": 1549365736923,
      "signature":
↪"4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CdddnyBW",
      "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
      "recipient": "3N9iRMou3pgmyPbFZn5QZQvBTQBkL2fR6R1",
      "amount": 1000000000
    }
```

(continues on next page)

(continued from previous page)

```
  ]
]
```

GET /transactions/unconfirmed

Returns all unconfirmed transactions from node utxpool.

Method Response:

```
[
  {
    "type": 4,
    "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
    "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
    "amount": 100000,
    "fee": 100000,
    "timestamp": 1549365736923,
    "attachment": "string",
    "signature":
↪"GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm"
  }
]
```

GET /transactions/unconfirmed/size

Return the number of transactions available in UTX pool.

GET /unconfirmed/info/{id}

Query transaction details from UTX pool by its ID.

POST /transactions/calculateFee

Calculates fee amount for transferred transaction.

Query Parameters

```
"type" - Transaction type
"senderPublicKey" - Public key of sender
"sender" is ignored
"fee" is ignored and all the other parameters appropriate for a transaction of the given type.
```

Method Query

```
{
 "type": 10,
 "timestamp": 1549365736923,
```

(continues on next page)

(continued from previous page)

```
 "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
 "alias": "ALIAS",
}
```

or

```
{
  "type": 4,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "recipient": "3P8JYPHrnXSfsWP1LVXySdzU1P83FE1ssDa",
  "amount": 1317209272,
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSzS3DfPuiXrURJ4AJS",
  "attachment": "string"
}
```

Method Response

```
{
  "feeAssetId": null,
  "feeAmount": 10000
}
```

or

```
{
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSzS3DfPuiXrURJ4AJS",
  "feeAmount": 10000
}
```

POST /transactions/sign

Signs a transaction with sender's private key stored in node keystore. After signing, method response must be sent to method input *Broadcast*.

It is necessary to enter the password into the `password` field in order to sign requests with the key from keystore node.

Sample queries

ID	Transaction type
3	*Issue*
4	*Transfer*
5	Reissue
6	Burn
7	Exchange
8	Lease
9	Lease Cancel
10	*Alias*
11	Mass Transfer
12	*Data*
13	*Set Script*
14	Sponsorship
101	Permission (for Genesis block)
102	*PermissionTransaction*
103	*CreateContractTransaction*
104	*CallContractTransaction*
105	*ExecutedContractTransaction*
106	*DisableContractTransaction*
107	*UpdateContractTransaction*
110	*GenesisRegisterNode Transaction*
111	*RegisterNode Transaction*
112	*CreatePolicy Transaction*
113	*UpdatePolicy Transaction*
114	*PolicyDataHash Transaction*
120	*AtomicTransaction Transaction*

3. Issue

```
{
   "type": 3,
   "version":2,
   "name": "Test Asset 1",
   "quantity": 100000000000,
   "description": "Some description",
   "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
   "decimals": 8,
   "reissuable": true,
   "fee": 100000000
}
```

4. Transfer

```
{
  "type": 4,
  "version": 2,
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 40000000000,
  "fee": 100000
}
```

10. Alias

```
{
   "type": 10,
   "version": 2,
   "fee": 100000,
   "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
   "alias": "hodler"
}
```

12. Data

```
{
   "type": 12,
   "version": 1,
   "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
   "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
   "author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
   "data":
   [
      {
      "key": "objectId",
      "type": "string",
      "value": "obj:123:1234"
      }
   ],
   "fee": 100000
}
```

13. Set Script

```
{
   "type": 13,
   "version": 1,
   "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
   "fee": 1000000,
   "name": "faucet",
   "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ=="
}

.. _tx-sponsorship:
```

14. Sponsorship

```
{
   "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
   "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
   "fee": 100000000,
   "isEnabled": false,
   "type": 14,
   "password": "1234",
   "version": 1
}
```

102. PermissionTransaction

Sample query

```
{
   "type":102,
```

(continues on next page)

(continued from previous page)

```
   "sender":"3NA9hBGoVPfJVybremiFgWN8REi9oiDydEF",
   "password": "",
   "fee": 1000000,
   "target":"3N8YKU9W1491pbcNnbKBpTSNCJmBaH2nbiT",
   "opType":"add",
   "role":"permissioner"
}
```

103. CreateContractTransaction

Sample query

```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1,
 }
```

Sample response

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

104. CallContractTransaction

Sample query

```
{
   "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
   "fee": 10,
   "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
   "type": 104,
   "version": 2,
   "contractVersion": 1
   "password": "",
   "params": [
      {
```

(continues on next page)

(continued from previous page)


```
            "type": "integer",
            "key": "a",
            "value": 1
      },
      {
            "type": "integer",
            "key": "b",
            "value": 100

      }
   ]
}
```


Sample response


```
{
   "type": 104,
   "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
   "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
   "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
   "fee": 10,
   "timestamp": 1549365736923,
   "proofs": [
      "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
   ],
   "version": 2,
   "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
   "params": [
      {
      "key": "a",
      "type": "integer",
      "value": 1
      },
      {
      "key": "b",
      "type": "integer",
      "value": 100
      }
   ]
}
```


105. ExecutedContractTransaction

Sample response


```
{
   "type": 105,
   "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
   "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
   "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
   "fee": 500000,
   "timestamp": 1549365523980,
   "proofs": [
      "4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
   ],
   "version": 1,
   "tx": {
```


(continues on next page)

(continued from previous page)


```
      "type": 103,
      "id": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
      "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
      "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
      "fee": 500000,
      "timestamp": 1549365501462,
      "proofs": [
         "2ZK1Y1ecfQXeWsS5sfcTLM5W1KA3kwi9Up2H7z3Q6yVzMeGxT9xWJT6jREQsmuDBcvk3DCCiWBdFHaxazU8pbo41"
      ],
      "version": 1,
      "image": "localhost:5000/contract256",
      "imageHash": "930d18dacb4f49e07e2637a62115510f045da55ca16b9c7c503486828641d662",
      "params": []
   },
   "results": []
}
```


106. DisableContractTransaction

Sample query

```
{
    "sender":"3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "password": "",
    "contractId":"Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
    "fee":500000,
    "type":106
}
```

Sample response


```
{
 "type": 106,
 "id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
 "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
 "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
 "fee": 500000,
 "proofs": [
↪"5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQSz75LAWix1ok2Z6PC5ezPpjqzqnr15i3RQmaEc" ],
 "version": 1,
 "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
 "height": 1632
}
```


107. UpdateContractTransaction

Sample query

```
{
  "image" : "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender" : "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "",
  "fee" : 100000000,
  "contractId" : "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
  "imageHash" : "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
  "type" : 107,
  "version" : 1
 }
```

Sample response


```
{
  "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "tx":
  {
  "senderPublicKey":
↪"5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "image":"registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender":"3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "proofs":[
↪"3tNsTyteeZrxEbVSv5zPT6dr247nXsVWR5v7Khx8spypgZQUdorCQZV2guTomutUTcyxhJUjNkQW4VmSgbCtgm1Z"],
  "fee":0,
  "contractId":"EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRCWA",
  "id":"HdZdhXVveMT1vYzGTviCoGQU3aH6ZS3YtFpYujWeGCH6",
  "imageHash":"17d72ca20bf9393eb4f4496fa2b8aa002e851908b77af1d5db6abc9b8eae0217",
  "type":107,"version":1,"timestamp":1572355661572},
  "sender":"3HfRBedCpWi3vEzFSKEZDFXkyNWbWLWQmmG",
  "proofs":[
↪"28ADV8miUVN5EFjhqeFj6MADSXYjbxA3TsxSwFVs18jXAsHVaBczvnyoUSaYJsjRNmaWgXbpbduccRxpKGTs6tro"],
  "fee":0,"id":"7niVY8mjzeKqLBePvhTxFRfLu7BmcwVfqaqtbWAN8AA2",
  "type":105,
  "version":1,
  "results":[],
  "timestamp":1572355666866
  }
 }
```


110. GenesisRegisterNode

Sample query


```
{
   "type": 110,
   "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "fee": 0,
   "timestamp": 1489352400000,
   "signature":
↪"2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "targetPublicKey": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
   "target": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj"
}
```


Sample response


```
{
   "signature":
↪"2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "fee": 0,
   "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkyMtmJMgYLpvwynbxHPTFpFEfFdyLpJ",
   "type": 110,
   "targetPublicKey": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
   "timestamp": 1489352400000,
   "target": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
   "height": 1
}
```


111. RegisterNode

Sample query

```
{
 "type": 111,
 "opType": "add",
 "sender":"3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
 "password": "",
 "targetPubKey": "apgJP9atQccdBPAgJPwH3NBVqYXrapgJP9atQccdBPAgJPwHapgJP9atQccdBPAgJPwHDKkh6A8",
 "nodeName": "Node #1",
 "fee": 500000,
 }
```

112. CreatePolicy

Sample query

```
{
   "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
   "policyName": "Policy# 7777",
   "password":"sfgKYBFCF@#$fsdf()*%",
   "recipients": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
         "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
         "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
   ],
   "fee": 15000000,
   "description": "Buy bitcoin by 1c",
   "owners": [
     "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
     "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
     "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
   ],
   "type": 112
 }
```

113. UpdatePolicy

Sample query

```
{
  "policyId": "7wphGbhqbmUgzuN5wzgqwqtViTiMdFezSa11fxRV58Lm",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "proofs": [],
  "recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
      "3NwJfjG5RpaDfxEhkwXgwD7oX21NMFCxJHL"
   ],
   "fee": 15000000,
   "opType": "add",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
```

(continues on next page)

(continued from previous page)

```
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 113,
 }
```

114. PolicyDataHash

When a user sends confidential data to the network using the *POST /privacy/sendData* method, the node automatically generates the 114 transaction. **120. AtomicTransaction**

Sample query

```
{'sender': '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP',
'transactions': [
    {'senderPublicKey':
↪'5nGi8XoiGjjyjbPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFwFXJYUmto5Joox76c3pp2Mr1LjgodUJuxryCJofQ',
↪'amount': 10, 'fee':    10000000, 'type': 4, 'version': 3, 'atomicBadge': {'trustedSender':
↪'3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP'}, 'attachment': '', 'sender':
↪'3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c', 'feeAssetId': None, 'proofs':   [
↪'XQ7iAqkarmm14AATc2Y9cR3Z9WnirsH4kH6RUL4QdT82rEwsmWBbBfWrADLE9o4cp2VR39W6b3vdrwFgg1dX7m3'],
↪'assetId': None, 'recipient':    '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'id':
↪'FZ59wAZnkFUqXjn61vvyj59fRa3cuS6nzuW3vqoRMsM5', 'timestamp': 1602857131666},  {'senderPublicKey
↪': '56rV5kcR9SBsxQ9LtNrmp6V72S4BDkZUJaA6ujZswDneDmCTmeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj',
↪'amount': 20, 'fee':     10000000, 'type': 4, 'version': 3, 'atomicBadge': {'trustedSender':
↪'3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP'}, 'attachment': '', 'sender':
↪'3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'feeAssetId': None, 'proofs':    [
↪'5KaXUFan2JD6VsJeGNyBCXEwqCjUF1nASAzxjnPZzBydXA5RJyXQGaL6N9MQ8GDNori1nXw5FsDLBqc3CPM3ezsk'],
↪'assetId': None, 'recipient':    '3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c', 'id':
↪'8GTqE1cc6zTVxYgQxgHJWJitVsDFRc6GmU5FJcnp5gu2', 'timestamp': 1602857132314}
    ],
'type': 120,
'version': 1}
```

POST /transactions/broadcast

Sends a signed transaction to blockchain.

Method Query

```
{
 "type":10,
 "senderPublicKey":"G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
 "fee":100000000,
 "timestamp":1550591678479,
 "signature":
↪"4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSqxKWfEPJG",
 "alias":"dajzmj6gfuzmbfnhamsbuxivc"
}
```

Method Response

```
{
 "type":10,
 "id":"9q7X84wFuVvKqRdDQeWbtBmpsHt9SXFbvPPtUuKBVxxr",
 "sender":"3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
```

(continues on next page)

(continued from previous page)

```
 "senderPublicKey":"G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
 "fee":100000000,
 "timestamp":1550591678479,
 "signature":
↪"4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSqxKWfEPJG",
 "alias":"dajzmj6gfuzmbfnhamsbuxivc"
}
```

POST /transactions/signAndBroadcast

Signs and sends a signed transaction to the blockchain.

Method Query

```
{
"sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
"policyName": "Policy# 7777",
"password":"sfgKYBFCF@#$fsdf()*%",
"recipients": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
      "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
      "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
],
"fee": 15000000,
"description": "Buy bitcoin by 1c",
"owners": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
],
"type": 112
}
```

Method Response

```
{
  "senderPublicKey": "3X6Qb6p96dY4drVt3x4XyHKCRvree4QDqNZyDWHzjJ79",
  "policyName": "Policy for sponsored v1",
  "fee": 100000000,
  "description": "Privacy for sponsored",
  "owners": [
     "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
     "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t"
  ],
  "type": 112,
  "version": 2,
  "sender": "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
  "feeAssetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
  "proofs": [
     "3vDVjp6UJeN9ahtNcQWt5WDVqC9KqdEsrr9HTToHfoXFd1HtVwnUPPtJKM8tAsCtby81XYQReLj33hLEZ8qbGA3V"
  ],
  "recipients": [
     "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
```

(continues on next page)

(continued from previous page)

```
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t"
  ],
  "id": "EyymzQcM2LrsgGDFFxeGn8DhahJbFYmorcBrEh8phv5S",
  "timestamp": 1585307711344
}
```

Utils

Hint: The rules for generating queries to the node are given in module *How to use the REST API*.

POST /utils/hash/secure

Returns secure (double) hash of specified message.

Method query:

```
ridethewaves!
```

Method response:

```
{
  "message": "ridethewaves!",
  "hash": "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}
```

POST /utils/hash/fast

Returns hash of specified message.

Method query:

```
ridethewaves!
```

Method response:

```
{
  "message": "ridethewaves!",
  "hash": "DJ35ymschUFDmqCnDJewjcnVExVkWgX7mJDXhFy9X8oQ"
}
```

POST /utils/script/compile

Response parameters:

```
"script" - Base64 script
"complexity" - script complexity
"extraFee" - the fee for outgoing transactions set by the script
```

Method query:

```
let x = 1
(x + 1) == 2
```

Method response:

```
{
  "script":
↪"3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
↪",
  "complexity": 11,
  "extraFee": 10001
}
```

or

Method query:

```
x == 1
```

Method response:

```
{
  "error": "Typecheck failed: A definition of 'x' is not found"
}
```

POST /utils/script/estimate

Decoding base64 script.

Method query:

```
AQQAAAABeAAAAAAAAAAAAQkAAAAAAAACCQAAZAAAAAIFAAAAAXgAAAAAAAAAAAEAAAAAAAAAAAJdecYi
```

Method response:

```
{
  "script":
↪"3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFgzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
↪",
  "scriptText": "FUNCTION_CALL(FunctionHeader(==,List(LONG, LONG)),List(CONST_LONG(1), CONST_
↪LONG(2)),BOOLEAN)",
  "complexity": 11,
  "extraFee": 10001
}
```

GET /utils/time

Returns current node time.

Method response:

```
{
   "system": 1544715343390,
   "NTP": 1544715343390
}
```

POST /utils/reloadwallet

Reloads node keystore. Runs if new key pair was created in keystore without restarting node.

Method response:

```
{
   "message": "Wallet reloaded successfully"
}
```

21.2.2 Authorization service REST API methods

You can read more about working with REST API in *this* section. The authorization service REST API methods are accessed via HTTPS protocol. Methods are closed by authorization and are marked with the  icon.

Ways of authorization

Depending on a used authorization method different values for access to the node REST API are set.

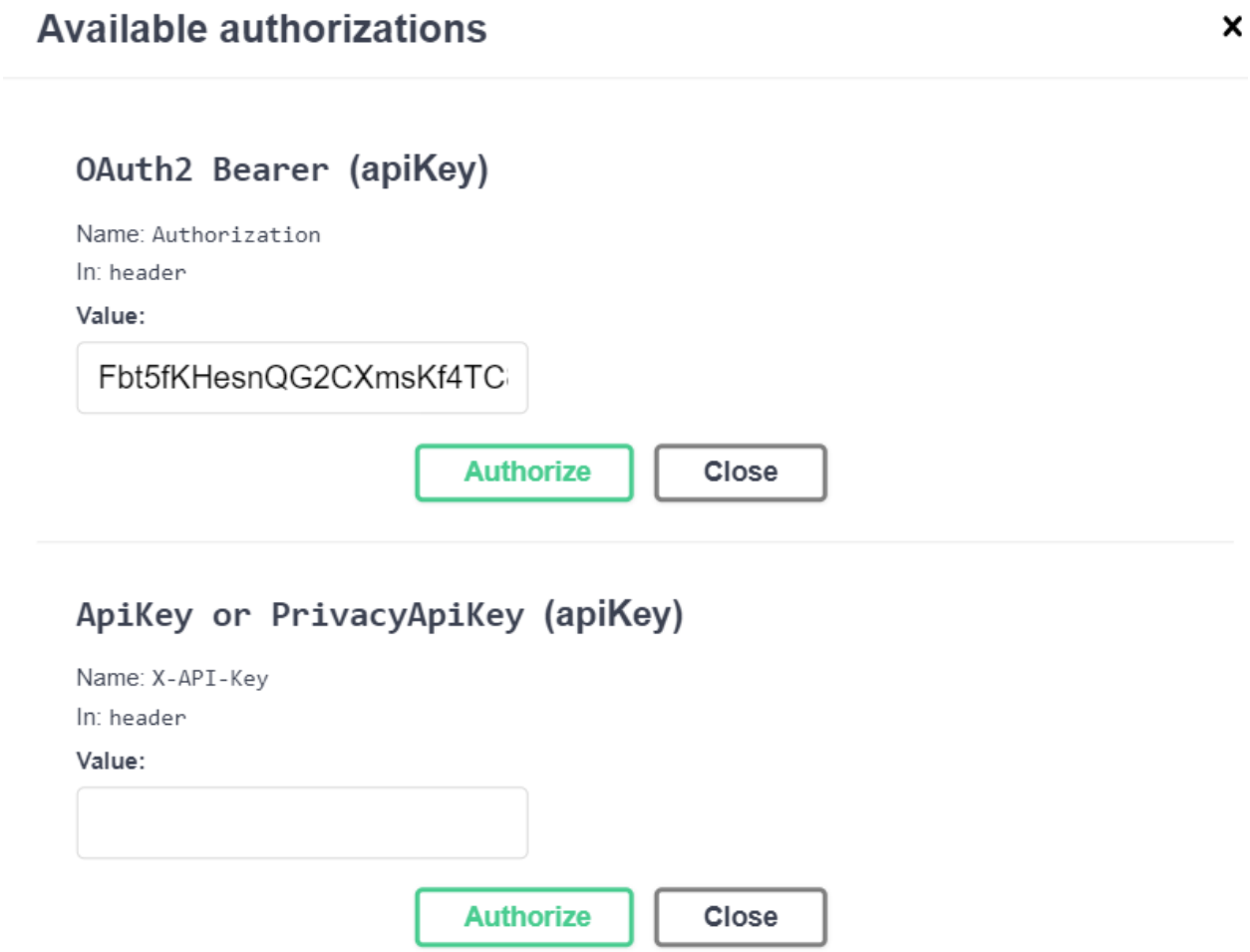
- `OAuth2 Bearer (apiKey)` **access** value of the token.
- `ApiKey or PrivacyApiKey (apiKey)` **apikeyhash** value for shared access to the node REST API, as well as for access to the *privacy* methods.

`Apikeyhash` authorization

Generation of an `apikeyhash` query is set during the *node configuration*. The value of the `restapi.apikeyhash` field can be also obtained with the use of the */utils/hash/secure* method of the node REST API. In order to sign queries with a node keystore key, set a keystore password in the `password` field of the `POST /transaction/sign` query.

Query example:

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'X-API-Key: 1' -d '1' 'http://2.testnet-pos.com:6862/transactions/calculateFee'
```


Available authorizations
OAuth2 Bearer (apiKey)
Name: Authorization
In: header
Value:
Fbt5fKHesnQG2CXmsKf4TC
Authorize
Close
ApiKey or PrivacyApiKey (apiKey)
Name: X-API-Key
In: header
Value:
Authorize
Close

Token authorization

If the *authorization service* is used, the client receives a pair of tokens, **refresh** and **access** for access to the node. Tokens can be obtained via the REST API of the authorization service.

The *POST /v1/user* method is used for registration of the user. Following parameters are passed to the input:

- `login` user login (email). A user email is used as a login.
- `password` account password.
- `locale` language of emails. Possible variants: *en* and *ru*.
- `source` user type. Possible variants: *license* and *voting*.

A user receives tokens only after registration.

In order to obtain and update tokens, following methods are used:

1. *POST /v1/auth/login* obtaining of an authorization token with the use of login and password. This method is used for authorization of users.
2. *POST /v1/auth/token* obtaining of **refresh** and **access** authorization tokens for services and applications. The method does not require parameters, and in reply sends token values. The method can be used only by administrators of authorization service.
3. *POST /v1/auth/refresh* updating of the **refresh** token. Token value is passed for input.

Authorization service methods

GET /status

Getting the authorization service status.

Method answer

```
{
 "status": "string",
 "version": "string",
 "commit": "string"
}
```

POST /v1/user

Registering a new user.

Method request

```
{
  "username": "string",
  "password": "string",
  "locale": "string",
  "source": "string"
 }
```

If the registration was successful, the response code is 201. Otherwise, the registration have failed.

GET /v1/user/profile

Getting user data.

Method answer

```
{
 "id": "string",
 "name": "string",
 "locale": "en",
 "addresses": [
 "string"
 ],
 "roles": [
 "string"
 ]
 }
```

POST /v1/user/address

Getting an user address.

Method request

```
{
  "address": "string",
  "type": "string"
 }
```

Method answer

```
{
  "addressId": "string"
 }
```

GET /v1/user/address/exists

Checking the user's email address. The method accepts the user's email address as an input parameter.

Method answer

```
{
   "exist": true
 }
```

POST /v1/user/password/restore

Restoring an user account password.

Method request

```
{
  "email": "string",
  "source": "string"
 }
```

Method answer

```
{
  "email": "string"
 }
```

POST /v1/user/password/reset

Reseting an user password.

Method request

```
{
  "token": "string",
  "password": "string"
 }
```

Method answer

```
{
  "userId": "string"
 }
```

GET /v1/user/confirm/{code}

Entering a confirmation code to reset an user account password. The value of the confirmation code is passed to the method as input.

POST /v1/user/resendEmail

Resending a password recovery code to the specified email address.

Method request

```
{
   "email": "string",
   "source": "string"
}
```

Method answer

```
{
   "email": "string"
}
```

POST /v1/auth/login

Registering a new user in the authorization service.

Method request

```
{
  "username": "string",
  "password": "string",
  "locale": "string",
  "source": "string"
 }
```

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
 }
```

POST /v1/auth/token

Registering external services and applications in the authorization service. This method does not require any request parameters.

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
 }
```

POST /v1/auth/refresh

Getting a new refresh token.

Method request

```
{
  "token": "string"
 }
```

Method answer

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
 }
```

GET /v1/auth/publicKey

Getting the authorization service public key.

Method answer

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA7d9Oj/ZQTkkjf4UuMfUu
QIFDTYxYf6QBKMVJnq/wXyPYYkV8HVFYFizCaEciv3CXmBH77sXnuTlrEtvK7zHB
KvV87OHmZuazjIgZVSkOnOY7F8UUVNXnlzVD1dPsOGJ6orM41DnC1W65mCrP3bjn
fV4RbmykN/lk7McA6EsMcLEGbKkFhmeq2Nk4hn2CQvoTkupJUnOCP1dhO4bq1lQ7
Ffj9K/FJq73wSXDoH+qqdRG9sfrtgrhtJHerruhv3456e0zyAcDO8+sJUQFKY8OB
SZMEndVzFS2ub9Q8e7BfcNxTmQPM4PhHO5wuTqL32qt3uJBx2OI4lu3OND44ZrDJ
BbVog73oPjRYXj+kTbwUZI66SP4aLcQ8sypQyLwqKk5DtLRozSNOOIrupJJ/pwZs
9zPEggL91TOrirbEhGlf5U8/6XN8GVXX4iMk2fD8FHLFJuXCD7Oj4JC2iWfFDC6a
uUkwUfqfjJB8BzIHkncoqOZbpidEE2lTWl+svuEu/wyP5rNlyMiE/e/fZQqM2+oO
cH5Qow6HH35BrloCSZciutUcd1U7YPqESJ5tryy1xn9bsMb+On1ocZTtvec/ow4M
RmnJwmOj1nd+cc19OKLG5/boeA+2zqWuOjCbWR9c0oCmgbhuqZCHaHTBEAKDWcsC
VRz5qD6FPpePpTQDb6ss3bkCAwEAAQ==
-----END PUBLIC KEY-----
```

21.2.3 REST API methods for the data service

Transactions

Set of methods that allows to show lists of transactions according to the parameters and filters being set.

GET /transactions

Returns a list of transactions matching the search query criteria and filters applied.

Important: It is returned a maximum of 500 transactions for the API **GET /transactions** method request.

Method Response:

```
[
 {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
```

(continues on next page)

(continued from previous page)

```
    "timestamp": 0,
    "version": 0
 }
]
```

GET /transactions/count

Returns the number of transactions matching the search query criteria and filters applied.

Method Response:

```
{
"count": "string"
}
```

GET /transactions/id/{id}

Returns transaction by ID {id}.

Method Response:

```
{
"id": "string",
"type": 0,
"height": 0,
"fee": 0,
"sender": "string",
"senderPublicKey": "string",
"signature": "string",
"timestamp": 0,
"version": 0
}
```

Token assets

Set of methods that allows to show information about token assets available in the blockchain.

GET /assets

Returns a list of token assets available in the blockchain (as token issue transactions).

Method Response:

```
[
  {
    "index": 0,
    "id": "string",
    "name": "string",
    "description": "string",
    "reissuable": true,
    "quantity": 0,
```

(continues on next page)

(continued from previous page)

```
    "decimals": 0
  }
]
```

POST /assets/count

Returns a list of token assets available in the blockchain.

Method Response:

```
{
    "count": 0
}
```

GET /assets/{id}

Returns information about the user as per user's {id}.

Method Response:

```
{
  "index": 0,
  "id": "string",
  "name": "string",
  "description": "string",
  "reissuable": true,
  "quantity": 0,
  "decimals": 0
}
```

Users

Set of methods that allows to show information about users (blockchain participants) according to parameters and filters being set.

GET /users

Returns a list of users matching the search query criteria and filters applied.

Method Response:

```
[
  {
    "address": "string",
    "aliases": [
      "string"
    ],
    "registration_date": "string",
    "permissions": [
      "string"
    ]
```

(continues on next page)

(continued from previous page)

```
  }
]
```

GET /users/count

Returns a list of users matching the search query criteria and filters applied.

Method Response:

```
{
  "count": 0
}
```

GET /users/{userAddressOrAlias}

Returns information about the user as per user's address or alias.

Method Response:

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```

GET /users/contractid/{contractId}

Returns a list of users that have ever called a smart contract with a specified {contractId}.

Method Response:

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```

POST /users/byaddresses

Returns a list of users for a specified set of addresses.

Method Response:

```
[
  {
    "index": 0,
    "id": "string",
    "name": "string",
    "description": "string",
    "reissuable": true,
    "quantity": 0,
    "decimals": 0
  }
]
```

Blocks

GET /blocks/at/{height}

Returns the block at the specified height.

Method Response:

```
{
  "version": 0,
  "timestamp": 0,
  "reference": "string",
  "features": [
    0
  ],
  "generator": "string",
  "signature": "string",
  "blocksize": 0,
  "transactionsCount": 0,
  "fee": 0,
  "height": 0,
  "transactions": [
    {
      "id": "string",
      "type": 0,
      "height": 0,
      "fee": 0,
      "sender": "string",
      "senderPublicKey": "string",
      "signature": "string",
      "timestamp": 0,
      "version": 0
    }
  ]
```

Smart contracts

Set of methods that allows to show information on Docker smartcontracts available in the blockchain.

GET /contracts

Returns a list of smart contracts in the blockchain according to the specified parameters and filters.

Method Response:

```
[
  {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0
  }
]
```

GET /contracts/count

Returns a number of transactions matching the search query criteria and filters applied.

Method Response:

```
{
  "count": 0
}
```

GET /transactions/id/{id}

Returns information about a smart contract according to its {id}.

Note: This method is able to return the smart contract state with the use of the `/state` endpoint. In order to return the state in a correct way, the method has to process the `lastIndex` and `limit` parameters to limit the number of entries to be returned. Example for a curl query: `curl X GET "https://<youraddress>/dataServiceAddress/contracts/id/{id}/state?lastIndex=0&limit=50&q="`

Method Response:

```
{
  "id": "string",
  "type": 0,
  "height": 0,
  "fee": 0,
  "sender": "string",
  "senderPublicKey": "string",
```

(continues on next page)

(continued from previous page)

```
  "signature": "string",
  "timestamp": 0,
  "version": 0
}
```

GET /contracts/id/{id}/versions

Returns a version history of a smart contract according to its {id}.

Method Response:

```
[
  {
    "version": 0,
    "image": "string",
    "imageHash": "string",
    "timestamp": "string"
  }
]
```

GET /contacts/history/{id}/key/{key}

Returns a history of changes of a smart contract key according to its {id} and {key}.

Method Response:

```
{
  "total": 777,
  "data": [
    {
      "key": "some_key",
      "type": "integer",
      "value": "777",
      "timestamp": 1559347200000,
      "height": 14024
    }
  ]
}
```

GET /contracts/senderscount

Returns a number of unique participants sending transactions 104 for smart contract call.

Method Response:

```
{
  "count": 777
}
```

GET /contracts/calls

Returns a list of 104 transactions with their parameters and results.

Method Response:


```
[
  {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
    "contract_id": "string",
    "contract_name": "string",
    "contract_version": "string",
    "image": "string",
    "fee_asset": "string",
    "finished": "string",
    "params": [
      {
        "tx_id": "string",
        "param_key": "string",
        "param_type": "string",
        "param_value_integer": 0,
        "param_value_boolean": true,
        "param_value_binary": "string",
        "param_value_string": "string",
        "position_in_tx": 0,
        "contract_id": "string",
        "sender": "string"
      }
    ],
    "results": [
      {
        "tx_id": "string",
        "result_key": "string",
        "result_type": "string",
        "result_value_integer": 0,
        "result_value_boolean": true,
        "result_value_binary": "string",
        "result_value_string": "string",
        "position_in_tx": 0,
        "contract_id": "string",
        "time_stamp": "string"
      }
    ]
  }
]
```

Access groups and privacy

Set of methods that allows to show information on access groups and nodes in the blockchain.

GET /privacy/groups

Returns a list of access groups in the blockchain.

Method Response:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/groups/count

Returns a number of access groups in the blockchain.

Method Response:

```
{
  "count": 0
}
```

GET /privacy/groups/{address}

Returns a list of access groups containing a specified {address}.

Method Response:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/groups/byrecipient/{address}

Returns a list of access groups that include a specified {address} as a recipient.

Method Response:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/groups/{address}/count

Returns a number of access groups that include a specified {address}.

Method Response:

```
{
  "count": 0
}
```

GET /privacy/groups/id/{id}

Returns information about the group according to its {id}.

Method Response:

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

GET /privacy/groups/id/{id}/history

Returns a history of changes of an access group according to its {id} (as a list of sent 112114 transactions with their descriptions).

Method Response:

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

GET /privacy/groups/id/{id}/history/count

Returns a number of sent 112114 transactions that have changed an access group with a specified {id}.

Method Response:

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

GET /privacy/nodes

Returns a list of nodes in the blockchain.

Method Response:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/nodes/count

Returns a number of available nodes in the blockchain.

Method Response:

```
{
  "count": 0
}
```

GET /privacy/nodes/publicKey/{targetPublicKey}

Returns information on a node according to its public key {targetPublicKey}.

Method Response:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/nodes/address/{address}

Returns information on a node according to its {address}.

Method Response:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

Data transactions

GET /api/v1/txIds/{key}

Returns a list of data transaction ID's containing the specified {key}.

Method Response:

```
[
  {
    "id": "string"
  }
]
```

GET /api/v1/txIds/{key}/{value}

Returns a list of data transaction ID's containing the specified {key} and {value}.

Method Response:

```
[
  {
        "id": "string"
  }
]
```

GET /api/v1/txData/{key}

Returns data transaction bodies containing the specified {key}.

Method Response:

```
[
  {
    "id": "string",
    "type": "string",
    "height": 0,
    "fee": 0,
```

(continues on next page)

(continued from previous page)


```
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
    "key": "string",
    "value": "string",
    "position_in_tx": 0
  }
]
```


GET /api/v1/txData/{key}/{value}

Returns data transaction bodies containing the specified {key} and {value}.

Method Response:


```
[
  {
    "id": "string",
    "type": "string",
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
    "key": "string",
    "value": "string",
    "position_in_tx": 0
  }
]
```


Leasing functions

GET /leasing/calc

Returns leasing payouts for a specified block heights.

Method Response:


```
{
  "payouts": [
    {
      "leaser": "3P1EiJnPxFxGyhN9sucXfB2rhQ1ws4cmuS5",
      "payout": 234689
    }
  ],
  "totalSum": 4400000,
  "totalBlocks": 1600
}
```

Data Service utilities

GET /info

Returns information about the data service in use.

Method Response:

```
{
  "version": "string",
  "buildId": "string",
  "gitCommit": "string"
}
```

GET /status

Returns information about the status of the data service.

Method Response:

```
{
  "status": "string"
}
```

Statistics and monitoring functions

GET /stats/transactions

Returns information on accomplished transactions within a specified time period.

Method Response:

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "transactions": [
        {
          "type": 104,
          "count": 100
        }
      ]
    }
  ]
}
```

GET /stats/contracts

Returns information on smart contract call transactions performed within a specified time period.

Method Response:


```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "transactions": [
        {
          "type": 104,
          "count": 100
        }
      ]
    }
  ]
}
```


GET /stats/tokens

Returns information on token turnover in the blockchain within a specified time period.

Method Response:


```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "sum": "12000.001"
    }
  ]
}
```


GET /stats/addressesactive

Returns addresses that have been active within a specified time period.

Method Response:


```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "senders": "12",
      "recipients": "12"
    }
  ]
}
```

GET /stats/addressestop

Returns addresses that have been the most active senders or recipients within a specified time period.

Method Response:


```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "senders": "12",
      "recipients": "12"
    }
  ]
}
```


GET /stats/nodestop

Returns node addresses that have created the most number of blocks within a specified time period.

Method Response:


```
{
  "limit": "10",
  "data": [
    {
      "generator": "3NdPsjaFC7NeioGVF6X4J5A8FVaxdtKvAba",
      "count": "120",
      "node_name": "Genesis Node #5"
    }
  ]
}
```


GET /stats/contractcalls

Returns a list of smart contracts that have been called the most number of times within a specified time period.

Method Response:


```
{
  "limit": "5",
  "data": [
    {
      "contract_id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "count": "120",
      "contract_name": "oracle_contract",
      "last_call": "60.321"
    }
  ]
}
```

GET /stats/contractlastcalls

Returns a list of last calls of smart contracts according to their id's and names.

Method Response:


```
{
  "limit": "5",
  "data": [
    {
      "contract_id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "contract_name": "oracle_contract",
      "last_call": "60.321"
    }
  ]
}
```


GET /stats/contracttypes

Returns a list of the blockchain smart contracts according to their image names and hashes.

Method Response:


```
{
  "limit": "5",
  "data": [
    {
      "id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "image": "registry.wvservices.com/waves-enterprise-public/oracle-contract:v0.1",
      "image_hash": "936f10207dee466d051fe09669d5688e817d7cdd81990a7e99f71c1f2546a660",
      "count": "60",
      "sum": "6000"
    }
  ]
}
```


GET /stats/monitoring

Returns information about the network.

Method Response:


```
{
  "tps": "5",
  "blockAvgSize": "341.391",
  "senders": "50",
  "nodes": "50",
  "blocks": "500000"
}
```

Anchoring functions

GET /anchoring/rounds

Returns a list of transactions matching the specified parameters and filters.

Method Response:

```
[
  {
    "height": 0,
    "sideChainTxIds": [
      "string"
    ],
    "mainNetTxIds": [
      "string"
    ],
    "status": "string",
    "errorCode": 0
  }
]
```

GET /anchoring/round/at/{height}

Returns information about an anchoring round on a specified block {height}.

Method Response:

```
{
  "height": 0,
  "sideChainTxIds": [
    "string"
  ],
  "mainNetTxIds": [
    "string"
  ],
  "status": "string",
  "errorCode": 0
}
```

GET /anchoring/info

Returns information about the anchoring in the blockchain.

Method Response:

```
{
  "height": 0,
  "sideChainTxIds": [
    "string"
  ],
  "mainNetTxIds": [
    "string"
  ],
  "status": "string",
```

(continues on next page)

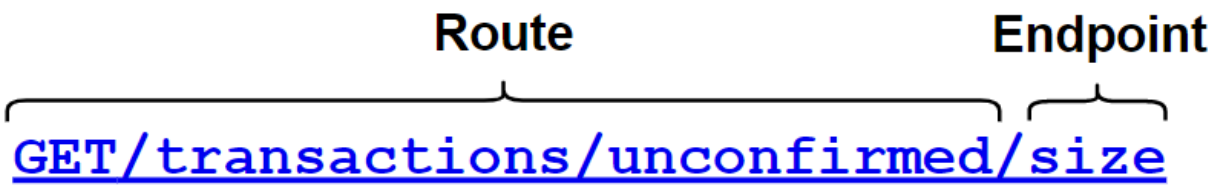
(continued from previous page)

```
  "errorCode": 0
}
```

21.2.4 How to use the REST API

All calls of API methods are GET, POST or DELETE https queries to the URL `https://yournetwork.com/nodeN/apidocs/swagger.json` with a definite set of parameters. Needed groups of queries are set in the Swagger interface, as well as routes with endpoints. A route in Swagger is a URL to an http method, and an endpoint is a final part of a route, the query to a method itself. Example:

URL to an HTTP method



The compulsory `apikeyhash` authorization is needed for queries that need the actions mentioned above. Authorization type is set in the node configuration file. If the `apikeyhash` authorization type has been set, a secret phrase value should be stated during authorization. A secret phrase hash is stated in the node configuration file (`restapi.apikeyhash` field).

- access to the keystore of the node (for instance, sign method);
- access to operations with groups of access to private data;
- access to the node configuration.

In case of token authorization, the token **access** value is set in a corresponding field. If the token authorization is set, all REST API methods for access to the node are closed.

CHAPTER

TWENTYTWO

DOCKER SMARTCONTRACTS

22.1 Preparing to work

In order to start working with containerized smart contracts, their execution should be set up.

Execution of smart contracts is set up in the node configuration file. There is also an opportunity to set up a used image for a separate smart contract with the use of the ref:*103 CreateContractTransaction <CreateContractTransaction>* transaction.

22.1.1 Setting up of Docker contracts execution in the node configuration file

This method allows to flexibly set up execution of all Docker contracts for the node. To do this, there is the `dockerengine` section in the `node.conf` file containing following configuration parameters:

- `enable` – enable processing of transactions for all Docker contracts.
- `integrationtestsmodeenable` – test mode for Docker contracts. With this option, smart contracts are executed locally in a container.
- `dockerhost` – docker daemon address (optional).
- `noderestapi` – path to the node REST API (optional).
- `startuptimeout` – time period for creation of a gRPC contract container and its registration in the node (in seconds).
- `timeout` – time period for contract execution (in seconds).
- `memory` – memory limit for a contract container (in megabytes).
- `memoryswap` – virtual memory for a contract container (in megabytes).
- `reusecontainers` – using of the same container for multiple contracts that use the same Docker image.
- `removecontainerafter` – the time interval of container inactivity, after which it will be removed.
- `allownetaccess` – allowing of access to the network.
- `remoteregistries` – address of Docker repositories and authorization settings for them.
- `checkregistryauthonstartup` – check of Docker repositories authorization on the node startup.
- `defaultregistrydomain` – default address of a Docker repository (optional). This parameter is used, if a repository is not stated in the contract image name.
- `contractexecutionmessagescache` – settings of the cache with statuses of execution of Docker contracts transactions;
- `expireafter` – time period for storage of a smart contract status.

- `maxbuffersize` and `maxbuffertime` – settings of volume and time for storage of status caches.
- `contractauthexpiresin` – lifetime of an authorization token used by smart contracts for node calls.
- `grpcserver` – section of gRPC server settings for interaction of Docker contracts with the gRPC API.
- `host` – node network address (optional).
- `port` – gRPC port.
- `akkahttpsettings` – section for settings of the Akka HTTP framework, used for the gRPC server.
- `removecontaineronfail` – removing of a container if an error has taken place during its start.

The `remoteregistries` parameters block can include addresses of multiple repositories. If this parameter is used, address of each repository should be stated for access to it, as well as username and password. In order to address to a definite repository from a list, the address of this repository should be stated in the Docker image name.

The `defaultregistrydomain` parameter is used in cases when a repository address is not stated in the Docker image name. If this parameter is used and the Docker image name includes a repository address, a smart contract refers to the repository stated in the image name.

Hint: If the repository path is stated with the use of the 103 `CreateContractTransaction` transaction, this path has a priority to the *remoteregistries`* and `defaultregistrydomain` node parameters.

Example of the `dockerengine` section of the node configuration file

In this example, we consider a variant of the Docker contract execution with the repository address and its authorization settings (the `remoteregistries` block is filled in, the `defaultregistrydomain` parameter is commented out). The standard Docker daemon and the node REST API are also used (the `dockerhost` and `noderestapi` parameters are commented out). Removal of a container in case of a startup error (parameter `removecontaineronfail`) is enabled to find errors while working with smart contracts.

```
docker-engine {
  enable = yes
  integration-tests-mode-enable = no
  # docker-host = "unix:///var/run/docker.sock"
  # node-rest-api = "https://restapi.clientservice.com/"
  execution-limits {
    startup-timeout = 10s
    timeout = 10s
    memory = 512
    memory-swap = 0
    }
  reuse-containers = yes
  remove-container-after = 10m
  allow-net-access = yes
  remote-registries = [
    {
     domain = "myregistry.com:5000"
     username = "user"
     password = "password"
   }
  ]
  check-registry-auth-on-startup = no
  # default-registry-domain = "registry.wavesenterprise.com"
```

(continues on next page)

(continued from previous page)

```
  contract-execution-messages-cache {
    expire-after = 60m
    max-buffer-size = 10
    max-buffer-time = 100ms
  }
  contract-auth-expires-in = 1m
  grpc-server {
    # host = "192.168.97.3"
    port = 6865
    akka-http-settings {
        akka {
        http.server.idle-timeout = infinite
        http.client.idle-timeout = infinite
        http.host-connection-pool.idle-timeout = infinite
        http.host-connection-pool.client.idle-timeout = infinite
        }
    }
  }
  remove-container-on-fail = yes
}
```

22.1.2 Setting up of definite Docker contract execution with the use of the 103 CreateContractTransaction

The *103 CreateContractTransaction* transaction is used for creation of a Docker contract. While developing this transaction, there is an opportunity to set an image for execution of a smart contract created with the use of this transaction. To do this, following parameters are used:

- `image` name of a Docker image for a smart contract to be created.
- `imagehash` hash sum of a used Docker image.
- `contractname` smart contract name.
- `password` password for access to a Docker repository (optional).

Hint: If only an image name is specified in the `image` field, the smart contract accesses the address specified in the `remoteregistries` or `defaultregistrydomain` node configuration parameters and finds the Docker image specified in the `image` field of the transaction at that address. This field may also contain the full image address: in this case, the image at that address is used to execute the smart contract being created, and the corresponding node configuration parameters are not applied.

Example of setting up of Docker contract execution with the use of the 103 `CreateContractTransaction`

In the above example, we consider a transaction that creates a Docker contract from a separate image whose full address is specified in the `image` field. Accordingly, the addresses specified in the node configuration file will not be applied to this smart contract.

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
```

(continues on next page)

(continued from previous page)

```
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "customregistry.com:5000/stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

22.2 Smart contract run with gRPC

In addition to using the REST API, smartcontracts can work with a node via the gRPC framework. A general technical description of the implementation features of contracts is given in the *Docker Smart Contracts* section. Before writing a smart contract, you need to perform the preparatory steps given in the section *How to use the gRPC framework*

In the following section there is an example of developing of a smart contract which increments a number (increases it by 1).

22.2.1 Description of the smart contract

In our example *103* transaction initializes the initial state of the contract for the creation, keeping the numeric key `sum` with `0` value in it:

```
{
    "key": "sum",
    "type": "integer",
    "value": 0
}
```

Each next *104* call transaction increases the key value `sum` by one (`sum = sum + 1`).

How the smart contract works after the call:

1. After the program runs, it checks for the presence of environment variables. There are environment variables which are used by the contract:

    - `CONNECTION_ID` – connection ID passed by the contract when connecting to a node.
    - `CONNECTION_TOKEN` – authorization token passed by the contract when connecting to a node.
    - `NODE` – a node IP address or a node domain name.
    - `NODE_PORT` – a gRPC port of the service which is deployed on the node.

    The values of the `NODE` and `NODE_PORT` variables are taken from :ref:`dockerengine.grpcserver <dockerconfiguration> section of the configuration file. Other variables are generated by the node and passed to the container when creating a smart contract.

2. Using `NODE` and `NODE_PORT` variables values the contract creates gRPC connection to a node.

3. Then gRPC `ContractService` service's `Connect` method is called (see additional info in the *contract.proto* file). This method accepts `ConnectionRequest` gRPC message which is specifying the connection ID (`CONNECTION_ID` environment variable). Also in the methods metadata you need to specify the `authorization` head which contains an authorization token (`CONNECTION_TOKEN` environment variable).
4. In the case of successful result gRPC `stream` is return including the `ContractTransactionResponse` objects for the execution. The `ContractTransactionResponse` object contains two fields:
    - `transaction` – a contract creation or call transaction.
    - `auth_token` – an authorization token, specified in the `authorization` head of metadata of gRPC method being called.

    If `transaction` contains a creation transaction (transaction type – *103*), the initial state is initialized for the contract. If `transaction` contains a call transaction (transaction type – *104*), the following actions are performed:
    - the node receives a request of the value of the `sum` key (the `GetContractKey` method of the `ContractService` service);
    - the key value increases by one, `sum = sum + 1`);
    - a new key value is saved on the node (the `CommitExecutionSuccess` method of the `ContractService` service), i.e. the contract state is updated.

22.2.2 Smart contract creation

1. Download and install Docker for Developers (https://www.docker.com/get-started) for your operating system.
2. Prepare an image of the contract. The contract folder must contain the following files:
    - `src/contract.py`
    - `Dockerfile`
    - `run.sh`
    - `src/protobuf/contract.proto`
    - `src/protobuf/common.proto`
    - `src/protobuf/common_pb2.py`
    - `src/protobuf/contract_pb2.py`
    - `src/protobuf/contract_pb2_grpc.py`

    `src/protobuf/common_pb2.py`, `src/protobuf/contract_pb2.py`, `src/protobuf/contract_pb2_grpc.py` files should be generated by the gRPC compiler using the `contract.proto` and `common.proto` protobuf files.

    Important: After compiling the files you need to change the `import` directive in the generated files:
    - it must be `import protobuf.common_pb2 as common__pb2` in the `contract_pb2.py` file;
    - it must be `import protobuf.contract_pb2 as contract__pb2` in the `contract_pb2_grpc.py` file.

    ---

3. If you want that your contracts transactions could be processed simultaneously, you should pass the `asyncfactor` parameter in the contract code itself. The contract passes the value of the `asyncfactor` parameter as part of the `Connection Request` gRPC message:

```
message ConnectionRequest {
string connection_id = 1;
int32 async_factor = 2;
}
```

The value of the `asyncfactor` parameter can be preset in the range from 1 to 999, or dynamically calculated. You can set a fixed value for this parameter as a constant, but we recommend setting the calculated value for this parameter. For example, a contract can request the number of available cores and pass this number as the value of the `asyncfactor` parameter. This number will be used for parallel processing of contracts transactions. If the `asyncfactor` parameter is not defined, then all contracts transactions will be processed sequentially by default.

Note that not all development tools can support parallel processing of contract code. Also, the logic of the contract code should take into account the specifics of parallel execution of the contract. For more information about parallel contract processing, see *Parallel contract execution*.

4. Install the image in the Docker image repository. If you are using a local repository, run the following commands in the terminal:

```
docker run -d -p 5000:5000 --name registry registry:2
cd contracts/grpc-increment-contract
docker build -t grpc-increment-contract .
docker image tag grpc-increment-contract localhost:5000/grpc-increment-contract
docker start registry
docker push localhost:5000/grpc-increment-contract
```

5. Use `docker inspect` command to get more info about smart contract:

```
docker inspect 57c2c2d2643d
[
{
"Id": "sha256:57c2c2d2643da042ef8dd80010632ffdd11e3d2e3f85c20c31dce838073614dd",
"RepoTags": [
    "wenode:latest"
],
"RepoDigests": [],
"Parent": "sha256:d91d2307057bf3bb5bd9d364f16cd3d7eda3b58edf2686e1944bcc7133f07913",
"Comment": "",
"Created": "2019-10-25T14:15:03.856072509Z",
"Container": "",
"ContainerConfig": {
    "Hostname": "",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
```

Important: The smart contract identifier `Id` is the value of the `imageHash` field and it is used in transactions with the created smart contract.

6. Sign the *103* transaction for the smart contract creation. In our example the transaction is signed with a key stored in the node's keystore. See REST API section for a description of the rest API nodes and

rules for generating transactions.

Request sample of the contract creation transaction:

```
{
    "fee": 100000000,
    "image": "localhost:5000/grpc-increment-contract",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "grpc-increment-contract",
    "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2",
    "password": "",
    "params": [],
    "type": 103,
    "version": 2,
}
```

Curlrequest sample:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-Api-Token' -d '{ \
    "fee": 100000000, \
    "image": "localhost:5000/grpc-increment-contract", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "grpc-increment-contract", \
    "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2", \
    "password": "", \
    "params": [], \
    "type": 103, \
    "version": 2 \
}' 'http://localhost:6862/transactions/sign'
```

Response sample:

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 100000000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 2,
    "image": "localhost:5000/grpc-increment-contract",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "grpc-increment-contract",
    "params": [],
    "height": 1619
}
```

7. Send the signed transaction to the blockchain. A response from the *sign* method should be passed to *broadcast* method input.

Request sample for sending a smart contract creation transaction to the blockchain:

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
```

(continues on next page)

(continued from previous page)

```
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
}
```

Curlrequest sample:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-Api-Token' -d '{ \
    "type": 103, \
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M", \
    "fee": 100000000, \
    "timestamp": 1550591678479, \
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ], \
    "version": 2, \
    "image": "localhost:5000/grpc-increment-contract", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "grpc-increment-contract", \
    "params": [], \
    "height": 1619 \
}' 'http://localhost:6862/transactions/broadcast'
```

Response sample:

```
{
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 100000000,
    "timestamp": 1550591678479,
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 2,
    "image": "localhost:5000/grpc-increment-contract",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "grpc-increment-contract",
    "params": [],
    "height": 1619
}
```

Compare transaction identifiers of both operations (`id` field) and make sure, that the initialization contract transaction has placed in the blockchain.

22.2.3 Smart contract call

1. Sign the *104* transaction for the smart contract call.

Request sample of the contract call transaction:

```
{
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "fee": 15000000,
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "password": "",
    "type": 104,
    "version": 2,
    "contractVersion": 1,
    "params": []
}
```

2. Send the signed transaction to the blockchain. A response from the *sign* method should be passed to *broadcast* method input.

Note: Parameters of the 104 transaction (blocks to be added to the `params` section) support 4 data types: *string*, *integer*, *boolean*, *binary*. Usage example of these data types while forming a smart contract call is stated below.

Request sample for sending a smart contract call transaction to the blockchain:

```
{
    "type": 104,
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 15000000,
    "timestamp": 1549365736923,
    "proofs": [
        "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
    ],
    "version": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params": [ {
    "type" : "string",
    "value" : "data",
    "key" : "action"
    }, {
    "type" : "integer",
    "value" : 3,
    "key" : "number"
    }, {
    "type" : "boolean",
    "value" : true,
    "key" : "isPositive"
    }, {
    "type" : "binary",
    "value" : "base64:daaa",
    "key" : "code"
    } ]
}
```

Curlrequest sample:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-Api-Token' -d '{ \
    "type": 104, \
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP", \
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58", \
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq", \
    "fee": 15000000, \
    "timestamp": 1549365736923, \
    "proofs": [ \
        "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"␣
↪\
    ], \
    "version": 1, \
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2", \
    "params": [] \
}' 'http://localhost:6862/transactions/broadcast'
```

Response sample:

```
[
    {
        "key": "sum",
        "type": "integer",
        "value": 2
    }
]
```

Use the smart contract identifier to get info about an execution result.

22.2.4 Files samples

run.sh listing:

```
#!/bin/sh

eval $SET_ENV_CMD
python contract.py
```

Dockerfile listing:

```
FROM python:3.8-slim-buster
RUN apt update && apt install -yq dnsutils
RUN pip3 install grpcio-tools
ADD src/contract.py /
ADD src/protobuf/common_pb2.py /protobuf/
ADD src/protobuf/contract_pb2.py /protobuf/
ADD src/protobuf/contract_pb2_grpc.py /protobuf/
ADD run.sh /
RUN chmod +x run.sh
ENTRYPOINT ["/run.sh"]
```

Python smart contract listing:

```
import grpc
import os
import sys

from protobuf import common_pb2, contract_pb2, contract_pb2_grpc

CreateContractTransactionType = 103
CallContractTransactionType = 104

AUTH_METADATA_KEY = "authorization"

class ContractHandler:
    def __init__(self, stub, connection_id):
        self.client = stub
        self.connection_id = connection_id
        return

    def start(self, connection_token):
        self.__connect(connection_token)

    def __connect(self, connection_token):
        request = contract_pb2.ConnectionRequest(
            connection_id=self.connection_id
        )
        metadata = [(AUTH_METADATA_KEY, connection_token)]
        for contract_transaction_response in self.client.Connect(request=request,␣
↪metadata=metadata):
            self.__process_connect_response(contract_transaction_response)

    def __process_connect_response(self, contract_transaction_response):
        print("receive: {}".format(contract_transaction_response))
        contract_transaction = contract_transaction_response.transaction
        if contract_transaction.type == CreateContractTransactionType:
            self.__handle_create_transaction(contract_transaction_response)
        elif contract_transaction.type == CallContractTransactionType:
            self.__handle_call_transaction(contract_transaction_response)
        else:
            print("Error: unknown transaction type '{}'".format(contract_transaction.type),␣
↪file=sys.stderr)

    def __handle_create_transaction(self, contract_transaction_response):
        create_transaction = contract_transaction_response.transaction
        request = contract_pb2.ExecutionSuccessRequest(
            tx_id=create_transaction.id,
        r   esults=[common_pb2.DataEntry(
                    key="sum",
                    int_value=0)]
        )
        metadata = [(AUTH_METADATA_KEY, contract_transaction_response.auth_token)]
        response = self.client.CommitExecutionSuccess(request=request, metadata=metadata)
        print("in create tx response '{}'".format(response))

    def __handle_call_transaction(self, contract_transaction_response):
        call_transaction = contract_transaction_response.transaction
        metadata = [(AUTH_METADATA_KEY, contract_transaction_response.auth_token)]

        contract_key_request = contract_pb2.ContractKeyRequest(
```

(continues on next page)

(continued from previous page)

```
            contract_id=call_transaction.contract_id,
            key="sum"
        )
        contract_key = self.client.GetContractKey(request=contract_key_request, metadata=metadata)
        old_value = contract_key.entry.int_value

        request = contract_pb2.ExecutionSuccessRequest(
            tx_id=call_transaction.id,
            results=[common_pb2.DataEntry(
                key="sum",
                int_value=old_value + 1)]
        )
        response = self.client.CommitExecutionSuccess(request=request, metadata=metadata)
        print("in call tx response '{}'".format(response))

def run(connection_id, node_host, node_port, connection_token):
    # NOTE(gRPC Python Team): .close() is possible on a channel and should be
    # used in circumstances in which the with statement does not fit the needs
    # of the code.
    with grpc.insecure_channel('{}:{}'.format(node_host, node_port)) as channel:
        stub = contract_pb2_grpc.ContractServiceStub(channel)
        handler = ContractHandler(stub, connection_id)
        handler.start(connection_token)

CONNECTION_ID_KEY = 'CONNECTION_ID'
CONNECTION_TOKEN_KEY = 'CONNECTION_TOKEN'
NODE_KEY = 'NODE'
NODE_PORT_KEY = 'NODE_PORT'

if __name__ == '__main__':
    if CONNECTION_ID_KEY not in os.environ:
        sys.exit("Connection id is not set")
    if CONNECTION_TOKEN_KEY not in os.environ:
        sys.exit("Connection token is not set")
    if NODE_KEY not in os.environ:
        sys.exit("Node host is not set")
    if NODE_PORT_KEY not in os.environ:
        sys.exit("Node port is not set")

    connection_id = os.environ['CONNECTION_ID']
    connection_token = os.environ['CONNECTION_TOKEN']
    node_host = os.environ['NODE']
    node_port = os.environ['NODE_PORT']

    run(connection_id, node_host, node_port, connection_token)
```

contract.proto listing:

```
syntax = "proto3";
package wavesenterprise;

option java_multiple_files = true;
option java_package = "com.wavesplatform.protobuf.service";
option csharp_namespace = "WavesEnterprise";

import "google/protobuf/wrappers.proto";
```

(continues on next page)

(continued from previous page)

```
import "common.proto";

service ContractService {

  rpc Connect (ConnectionRequest) returns (stream ContractTransactionResponse);

  rpc CommitExecutionSuccess (ExecutionSuccessRequest) returns (CommitExecutionResponse);

  rpc CommitExecutionError (ExecutionErrorRequest) returns (CommitExecutionResponse);

  rpc GetContractKeys (ContractKeysRequest) returns (ContractKeysResponse);

  rpc GetContractKey (ContractKeyRequest) returns (ContractKeyResponse);
}

message ConnectionRequest {
  string connection_id = 1;
}

message ContractTransactionResponse {
  ContractTransaction transaction = 1;
  string auth_token = 2;
}

message ContractTransaction {
  string id = 1;
  int32 type = 2;
  string sender = 3;
  string sender_public_key = 4;
  string contract_id = 5;
  repeated DataEntry params = 6;
  int64 fee = 7;
  int32 version = 8;
  bytes proofs = 9;
  int64 timestamp = 10;
  AssetId fee_asset_id = 11;

  oneof data {
    CreateContractTransactionData create_data = 20;
    CallContractTransactionData call_data = 21;
  }
}

message CreateContractTransactionData {
  string image = 1;
  string image_hash = 2;
  string contract_name = 3;
}

message CallContractTransactionData {
  int32 contract_version = 1;
}

message ExecutionSuccessRequest {
  string tx_id = 1;
  repeated DataEntry results = 2;
```

(continues on next page)

(continued from previous page)

```
}

message ExecutionErrorRequest {
  string tx_id = 1;
  string message = 2;
}

message CommitExecutionResponse {
}

message ContractKeysRequest {
  string contract_id = 1;
  google.protobuf.Int32Value limit = 2;
  google.protobuf.Int32Value offset = 3;
  google.protobuf.StringValue matches = 4;
  KeysFilter keys_filter = 5;
}

message KeysFilter {
  repeated string keys = 1;
}

message ContractKeysResponse {
  repeated DataEntry entries = 1;
}

message ContractKeyRequest {
  string contract_id = 1;
  string key = 2;
}

message ContractKeyResponse {
  DataEntry entry = 1;
}

message AssetId {
  string value = 1;
}
```

common.proto listing:

```
syntax = "proto3";
package wavesenterprise;

option java_multiple_files = true;
option java_package = "com.wavesplatform.protobuf.common";
option csharp_namespace = "WavesEnterprise";

message DataEntry {
  string key = 1;
  oneof value {
    int64 int_value = 10;
    bool bool_value = 11;
    bytes binary_value = 12;
    string string_value = 13;
  }
}
```

22.3 gRPC services available to smart contract

The general description of smart contract operation with the use of gRPC is stated in the section *Docker smart contracts with the use of gRPC*.

You can use the official GitHub page for to download all required protobuf files. The list of all files is as follows:

- `address.proto` addresses methods.
- `common.proto` a common file for proper work of others protobuf files.
- `crypto.proto` methods for working with data encryption.
- `permission.proto` permission methods.
- `pki.proto` PKI methods.
- `privacy.proto` privacy methods.
- `util.proto` methods for utility tools.
- `contract.proto` contract methods.

Every protobuf file (except `common.proto`) contains a set of small blocks (message) that include a set of keyvalue fields. A list of such blocks for each file is provided below.

address.proto

- `GetAddresses` geting all addresses of participants whose key pairs are stored in the node keystore.
- `GetAddressData` getting all data recorded to address account {address}.

contract.proto

- `Connect` connecting a contract to a node.
- `CommitExecutionSuccess` getting the result of successful contract execution and sending the results to the node.
- `CommitExecutionError` getting a contract execution error and sending the results to the node.
- `GetContractKeys` getting the contract result execution by its ID (contract creation transaction ID).
- `GetContractKey` getting a contract execution value by its ID (contract creation transaction ID) and key {key}.

crypto.proto

- `EncryptSeparate` data encryption separately for the each recipient with the unique key.
- `EncryptCommon` data encryption with a single CEK key for all recipients and the CEK wraps into a unique KEK for the each recipient.
- `Decrypt` data decryption. The decryption is available only if the message recipient's key is in the node's keystore.

permission.proto

- `GetPermissions` getting roles (permissions) assigned to specified address {address} which are valid at the moment.
- `GetPermissionsForAddresses` getting roles (permissions) assigned to specified address list which are valid at the moment.

pki.proto

- `Sign` a creation a detached digital signature for sent data.
- `Verify` check the detached digital signature for sent data.

privacy.proto

- `GetPolicyRecipients` getting all addresses of participants, signed to the access group {policyid}.
- `GetPolicyOwners` getting all addresses of owners, signed to the access group {policyid}.
- `GetPolicyHashes` getting the array of identified hashes which are written with association to the {policyid}.
- `GetPolicyItemData` getting the confidential data package by its identified hash.
- `GetPolicyItemInfo` getting the metadata for the confidential data package by the identified hash.

util.proto

- `GetNodeTime` obtaining of the node current time.

22.4 REST API methods available to smart contracts

Important: REST API methods for Docker smart contracts are are being phased out.

Docker containerbased smart contracts can use the *node REST API*. For a general tutorial on how to create smart contracts using the REST API, see the article Docker smart contracts with the use of the node REST API.

Not all REST API methods are available to Docker smart contract developers. Below is a list of REST API node methods that a smart contract can use directly from the Docker container.

'Addresses' methods

- *GET /addresses*
- *GET /addresses/publicKey/{publicKey}*
- *GET /addresses/balance/{address}*
- *GET /addresses/data/{address}*
- *GET /addresses/data/{address}/{key}*

'Crypto' methods

- *POST /crypto/encryptCommon*
- *POST /crypto/encryptSeparate*
- *POST /crypto/decrypt*

'Privacy' methods

- *GET /privacy/{policyid}/getData/{policyitemhash}*
- *GET /privacy/{policyid}/getInfo/{policyitemhash}*
- *GET /privacy/{policyid}/hashes*
- *GET /privacy/{policyid}/recipients*

'Transactions' methods

- *GET /transactions/info/{id}*
- *GET /transactions/address/{address}/limit/{limit}*

'Contracts' methods

In order to enhance performance, a smart contract can use the *Contracts* methods via a dedicated `/internal/contracts/` route, that are fully identical to the plain *Contracts* methods.

- *GET /internal/contracts/{contractId}/{key}*
- *GET /internal/contracts/executedtxfor/{id}*
- *GET /internal/contracts/{contractId}*
- *GET /internal/contracts*

PKI methods

- *PKI /verify*

22.4.1 Authorization of a Docker smart contract

In order to work with the *node REST API*, a smart contract requires authorization. To ensure a correct interaction of a Docker contract with the API methods, following actions should be performed:

1. Define following variables as environment variables of a Docker contract:
   - `NODE_API` URL to the *node REST API*.
   - `API_TOKEN` Docker contract authorization token.
   - `COMMAND` commands for creation and call of a Docker contract.
   - `TX` transaction required by a Docker contract for work (codes *103* *107*).
2. A Docker contract developer assigns an `API_TOKEN` value to the header of the `XContractApiToken` request. Node writes to the `API_TOKEN` variable a JWT authorization token during creation and execution of a contract.
3. A contract code should transfer a received token in the header of the (`XContractApiToken`) request in every access to the node API.

CHAPTER

TWENTYTHREE

ROLE MANAGEMENT

The list of possible roles in the blockchain platform is given in module *"Authorization of participants"*.

Important: The prerequisite for changing permissions of participants (adding or deleting roles) is the availability of the participant's private key with the "permissioner" role in the node keystore from which the query is made.

23.1 Option 1: through REST API

Participant permissions are managed by signing (sign method) and broadcasting (broadcast method) of permission transactions through *Node REST API*.

Query object for sign method:

```
{
  "type":102,
  "sender":3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG,
  "senderPublicKey":4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g,
  "fee":0,
  "proofs":[""],
  "target":3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL,
  "opType":"add",
  "role":"contract_developer",
  "dueTimestamp":null
}
```

Query fields:

- `type` the type of the transaction for permission management of participants (type = 102);
- `sender` the participant address with the permission to issue permission transactions;
- `proofs` the transaction signature;
- `target` the participant address, for which permissions are required to be assigned or deleted;
- `role` participant permissions to be assigned or removed. Possible values: "miner", "issuer", "dex", "permissioner", "blacklister", "banned", "contract_developer", "connection_manager";
- `opType` the type of the operation "add" (add permissions) or "remove" (delete permissions);
- `dueTimestamp` the permission validity date in the timestamp format. The field is optional.

The response from the node is transferred to the broadcast method.

23.2 Option 2: using the Generators utility

With the use of the Generators utility, the permission management process can be automated.

Example of console launching:

```
java -jar generators.jar GrantRolesApp [configfile]
```

Example of configuration:

```
permission-granter {
waves-crypto = no
chain-id = T
account = {
    addresses = [
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w"
    ]
    storage = ${user.home}"/node/keystore.dat"
    password = "some string as password"
}
send-to = [
    "devnet-aws-fr-2.we.wavesnodes.com:6864"
]
grants = [
    {
    address: "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w"
    assigns = [
        {
        permission = "miner",
        operation = "add",
        due-timestamp = 1527698744623
        },
        {
        permission = "issuer",
        operation = "add",
        due-timestamp = 1527699744623
        },
        {
        permission = "blacklister",
        operation = "add"
        },
        {
        permission = "permissioner",
        operation = "remove"
        }
    ]
    }
]
txs-per-bucket = 10
}
```

The field “duetimestamp” limits the role validity; Fields “nodes”, “roles” are mandatory.

If the node is already assigned any of the roles specified in the config, then the case is handled in accordance with the rules:

Current node status	Status received from transaction	Processing result
No role assigned	New role	Success role assigned
Role assigned without dueDate	Role with dueDate	Checking dueDate; if less than current, then IncorrectDatetime, otherwise Success role assigned with duedate
Role assigned with dueDate	Role with dueDate	Checking dueDate; if less than current, then IncorrectDatetime, otherwise Success updating dueDate
Role assigned with dueDate	Role without dueDate	Success role assigned without dueDate
Role assigned with/without dueDate	Role removal	Checking node address; if <> for genesis address, then Success role removed

CHAPTER

TWENTYFOUR

PARTICIPANTS CONNECTION TO THE NETWORK

The moment of the first node *running* is the beginning of the new blockchain net creation. You can create the blockchain net from the starting only one node, further you can add new nodes as required.

- *Connect* a new node into the existing network.
- *Delete* unnecessary nodes from the network.

24.1 Connection of a new node to the existing net

You can add new nodes into the net at any time. The configuration files setting is described in the section *Installing and running the Waves Enterprise platform*. Perform all these actions and *run* the node. The following steps are making:

1. The new node user gives the public key and the node description to the net administrator.
2. The network administrator (the node with "Connectionmanager" role) uses the received public key and description for the *111 RegisterNode* transaction creation with the `"opType": "add"` parameter.
3. Transaction falls to the block and further into the nodes states of network participants. As a result of the transaction among the stored data, each participant of the network stores the public key and the address of the new node.
4. If necessary, the network administrator can add additional roles to the new node using the transaction *102 Permit*.
5. The user *runs* the node.
6. After starting, the node sends *handshakemessage* with its public key to the participants from the "peers" list of its configuration file.
7. Network participants compare the public key from the *handshake message* and the key from transaction *111 RegisterNode* sent earlier by the network administrator. If the check is successful, the network participant updates its database and sends the Peers Message message to the network.
8. Having successfully connected, the new node synchronizes with the network and receives the address table of the network participants.

24.2 Deleting the node

1. The network administrator creates the *111 RegisterNode* transaction with the parameter `"opType": "remove"` and the public key of the removed node within.
2. This transaction is fell into the block and approved by other nodes.
3. After accepting the transaction the nodes find the public key specified in the transaction *111 RegisterNode* in their state and delete it from there.
4. Then nodes delete the network address of the removed node from the `network.knownpeers` of the node configuration file.

CHAPTER

TWENTYFIVE

CONFIDENTIAL DATA EXCHANGE

Before you can share the confidential data, you need to create access groups. Using transactions, you can *add* or *change* access groups to the confidential data.

25.1 Creation of the confidential data access group

The confidential data access group can be created by any network participant. You need to specify the range of participants, which will get the data. Then any of participant will perform the following actions:

1. The network participant, the future owner of the group, is creating the *112 CreatePolicy* with the following parameters:

- **sender** the public key of the access group creator.
- **description** description of the access group.
- **policyName** the name of the access group.
- **recipients** public keys of access group participants, that will have the access to the confidential data.
- **owners** public keys of access group participants, which, in addition to the data access, can change the lineup of the group participants.

2. This transaction is fell into the block and approved by other nodes.
3. After accepting the transaction the nodes which are the access group participants will get the access to the confidential data.

25.2 Changing the access group

Access groups can only be changed by their owners. The following actions are performed to change the list of participants in the access group:

1. The group owner creates the *113 UpdatePolicy* transaction with the following parameters:

- **policyId** identifier of the access group;
- **sender** the public key of the access group owner;
- **opType** the option of the adding (`add`) or the removing (`remove`) the group participants;
- **recipients** public keys of access group participants, which are added or removed from the access group;
- **owners** public keys of access group participants, which are added or removed from the access group.

2. This transaction is fell into the block and approved by other nodes.
3. After accepting the transaction the information about participants of the changed access group will update.

25.3 Exchanging the confidential data

Important: The size of the transferred data via API method *POST /privacy/sendData* to the network is up to 20 MB.

---

1. Using the API *POST /privacy/sendData* tool the client sends the data to the network (API parameters: sender, password, policy ID, data type, data information, data and hash).
2. Access group participants use the *GET /privacy/{policyId}/getData/{policyItemHash}* tool for getting information about data and its further download.

Follow these steps for the values creation of the `data` and `hash` fields:

1. Translate the data byte sequence into the **Base64** encoding.
2. Place the result of the data conversion to the `"data": "29sCt...RgdC60LL"` field of the API *POST /privacy/sendData*.
3. Specify the data hash sum according to the **SHA256** algorithm in the `"hash": "9wetTB... SU2zr1Uh"` field. You need to specify the hash result in the **Base58** encoding.
4. Send the data to the network by pressing the *Try it out!* button.
5. Node automatically will create the *114 PolicyDataHash* transaction as a result of the data sending.

CHAPTER

TWENTYSIX

DATA ENCRYPTION OPERATIONS

Symmetric CEK and KEK keys are used to encrypt/decrypt data. CEK (Content Encryption Key) is the key for the encrypting text data, KEK (Key Encryption Key) is the key for encrypting the CEK. The CEK key is generated by a node randomly using the appropriate hashing algorithms. The KEK key is generated by a node based on DiffieHellman algorithm, using public and private keys of sender and recipients, and is used to encrypt the CEK key.

The symmetric CEK key is unreachable and does not appear in the encryption process. It is transmitted from the sender to the recipient in the encrypted form (wrappedKey) via open communication channels along with the encrypted message. One of such channels can be a record to the blockchain — a DataTransaction or a smart contract state. The KEK key does not transmit from the sender to recipients, it is restored by the recipient based on its private key and the known public key of the sender (DiffieHellman key exchange algorithm).

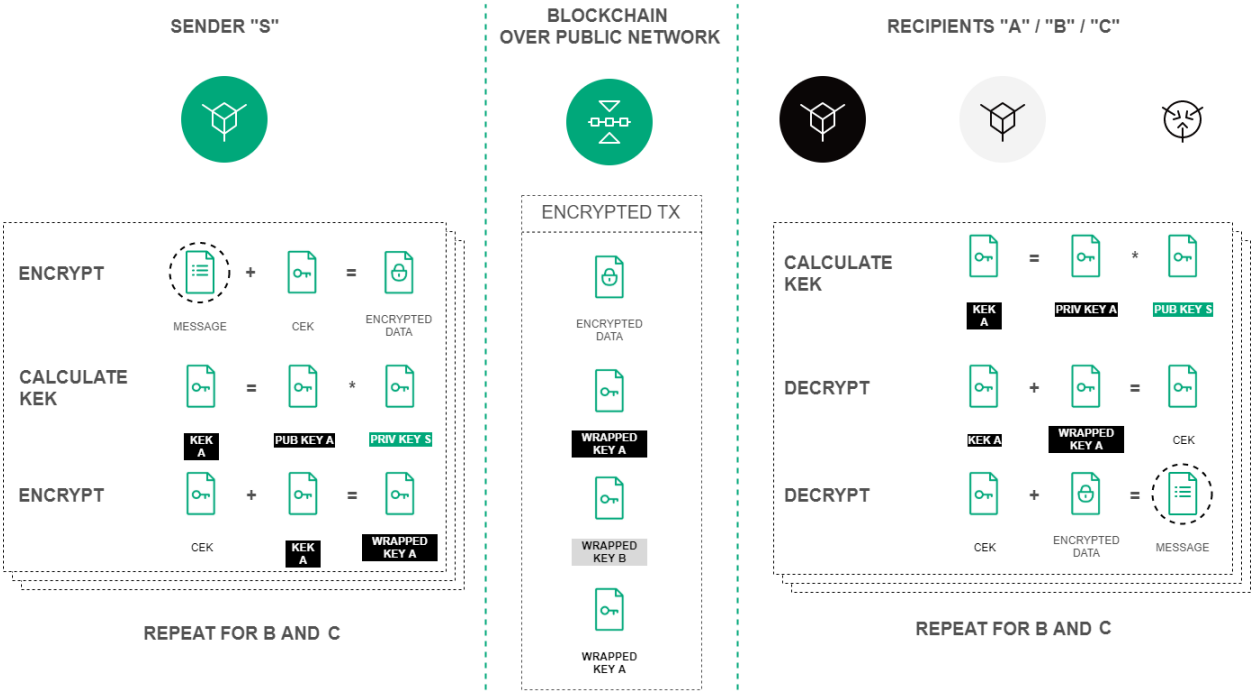


Fig. 1: Encryption procedure of the text data based on the DiffieHellman algorithm

Encryption/decryption process includes the following actions:

1. Use the *POST /crypto/encryptSeparate* method to encrypt data for each recipient separately. Parameters in the request object:

- `sender` the sender address;
- `password` a key pair password of the sender, which is generated at the same time as the account itself;
- `encryptionText` the text for the encryption;
- `recipientsPublicKeys` an array with recipients public keys list inside.

2. Use the *POST /crypto/encryptCommon* method to encrypt data for all recipients with a single CEK key.
3. Use the *POST /crypto/decrypt* method for the decryption. Parameters in the request object:

- `recipient` the recipient address.
- `password` a key pair password of the recipient, which is generated at the same time as the account itself.
- `encryptedText` the encrypted text data.
- `wrappedKey` the wrapped key obtained by encoding the data.
- `senderPublicKey` the sender public key.

CHAPTER

TWENTYSEVEN

JAVASCRIPT SDK

The JavaScript SDK is intended for apps developers that integrate with the Waves Enterprise platform. Using JavaScript SDK, users of developed applications sign and send all types of transactions to the blockchain network.

The JavaScript SDK uses REST API methods to work with the blockchain. Unlike direct interaction with the blockchain through a traditional REST API tool an application signs the transaction locally in the browser or in the Node.js environment without addressing to the node. The signed transaction is sent directly to the network. This way of working with the blockchain is more convenient and effective for developing thirdparty services and applications that interact with the blockchain network. Data is transmitted and received in *json* format over the HTTPS Protocol.

The JavaScript SDK works both in the browser and in the Node.js environment. Web applications use the browser option, and backend applications use the Node.js environment operation. If you want to work without a browser, you will need to install the LTS version to use the JavaScript SDK Node.js. The JavaScript SDK package itself and instructions for installing and initializing it can be found in GitHub. The General scheme of the JavaScript SDK is shown below.

27.1 JavaScript SDK kit

JavaScript SDK includes basic tools for work, as well as two auxiliary tools:

- **transactionsfactory** the component is intended for correct serialization into bytes of all types of transactions for subsequent signing with a private key.
- **signaturegenerator** a component that signs transactions and supports GOST and Wave *cryptography*.

27.2 Node cryptography methods in the JavaScript SDK

For realization of cryptographic algorithms, the SDK supports the *cryptographic methods of the node REST API*:

- crypto/encryptCommon
- crypto/encryptSeparate
- crypto/decrypt

27.3 Authorization in the blockchain via the JavaScript SDK

An application user applies standard authorization tools in the Waves Enterprise blockchain. For more information about authorization methods, see *authorization methods*.

The JavaScript SDK supports authorization in the browser and in the Node.js environment. The Fetch API interface is used for authorization in the browser. For Node.js the Axios HTTP client is used.

27.4 Signing and sending transactions to the blockchain

An app can create, sign, and send any transaction to the blockchain. See *transactions* section to learn more about transactions.

1. The creation of a transaction is initiated by an app.
2. All transaction fields are serialized to bytes and signed with the user's private key directly in the browser or in the Node.js environment.
3. The JavaScript SDK sends a transaction to the blockchain using the corresponding HTTP POST request.
4. An app receives a response in the form of a transaction hash to the executed HTTP POST request.

In order to obtain a transaction ID before its sending use a separate method:

```
const txHash = await Waves.API.Node.transactions.getTxId('transfer', txBody, { publicKey })
```

27.5 Creating a seed phrase

An app works with the seed phrase in the following ways:

- Creates a phrase in a completely random order.
- Creates an encrypted phrase with the specified password.
- Decrypts the phrase using the specified password.

CHAPTER

TWENTYEIGHT

GLOSSARY

Account

A client data set which is stored in database and used for client identification

Alias

A user's login associated with his address as a result of the transaction, the result of which is used to record the alias address matching in the database, and it is possible to specify this alias in the subsequent transactions

Anonymous network

Unpermissioned public blockchain which can be accessed by any participant as an anonymous person

Atomic container

A transaction that puts other transactions in a container that are executed atomically: either all of them are executed, or none of them are executed. The atomic transaction has *120* number

Blockchain

A decentralized, distributed and public digital ledger that is used to record in such way that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks

Genesis block

The first block in the blockchain which contains special genesis transactions distributing the initial balance and permissions

Access group

A table inside the node state containing the net participants list which can exchange the privacy data according to this policy

Cryptocurrency

A form of digital currency based on encryption algorithms and ran inside decentralized platforms built on the blockchain

Consensus

The way to agree on a single point of the data value in a network between participants

Mining

The process by which transactions are verified and added to a blockchain

Mainnet

A real network where transactions are executing, tokens are issuing and storing

Node

A computer which is ran the node software and connected to the blockchain network

Peer

A net address of the node

Private key

A privately held string of data that allows you to sign transactions and to get access to tokens. The private key is inextricably bound to the public key

Public network

Permissioned public blockchain where each participant is known and registered in the network

Public key

A string of data bound with the private key and used for interactions with net participants. The public key is applied to transactions to confirm the correctness of the user's signature made on the private key

Public address

A public address is the cryptographic hash of a public key and a net byte. They act as email addresses that can be published anywhere, unlike private keys

Rollback

Rollback of a created block for remining

Fork

Creation of a new blockchain branch

Swagger

API tool

Seed phrase

A set from 24 accidentally chosen words for restoring the access to the tokens

Smart account

An account with specified features for creating and running smartcontracts

Smart asset

A token with an attached script, during each new transaction with such a token the transaction will be confirmed first by the script, then by the blockchain

Smart contract

A computer program code that is capable of facilitating, executing, and enforcing the negotiation or performance of an agreement between participant

State

The full history of transactions which is stored in the node DB

Smart contract state

All current smart contract data, after each call 104 transaction data is updated

Address state

A complex entity that includes information related to any participant (blockchain address) balances, information about data transactions (in keyvalue format), smart contracts data (in keyvalue format) , etc.

Token

An account unit, a blockchain asset, which is not a cryptocurrency and is intended to represent the digital balance, it is an equivalent of the company's shares

Transaction

An operation that participants on the blockchain network use to interact with eachother

Participant

A blockchain participant who send transactions to the net for getting approve

Hash

A unique configuration of the symbols (letters and digits), it is a result of the hash function performing over the data according with the specified algorithm. Hash uniquely identifies the object

Private network

Permissioned private blockchain where all transactions are controlled by a central authority

Gateway

The app for tokens transfer from one blockchain net to another one

Airdrop

A distribution of cryptocurrency to users, entirely for free

CFT (Crash Fault Tolerance)

PoAbased consensus algorithm which is able to minimize creation of blockchain forks in case of any fault on a participant side.

PoS (ProofofStake)

A consensus algorithm based on the stake which is used for choosing the node for checking transactions and generating a new block

LPoS

Consensus Algorithm, which is an improved version of ProofofStake. A feature of the algorithm is that it provides the ability to lease the user's tokens

PoA (ProofofAuthority)

A consensus algorithm in a private blockchain that grants to the most authority nodes the right to check transactions and generate a new block

CHAPTER

TWENTYNINE

WHAT IS NEW IN THE WAVES ENTERPRISE

29.1 1.5.2

The 1.5.2 version is the latest released version and has the *latest* tag in the help system.

The page *CFT consensus algorithm* has been changed.

Version 1.5.2 contains critical updates, see the details in the release description.

29.2 1.5.0

The following pages have been added:

- *CFT consensus algorithm*
- *Preparing to work*
- gRPC methods of the node
- Tracking of blockchain events with the use of the gRPC interface

The following sections have been changed:

- *Cryptography*
- *Managing permissions*
- *Transactions*
- *Preparation of configuration files*
- *Changes in the node configuration file*
- *Description of the node configuration file parameters and sections*
- *Consensus configuration*
- Node API tools
- *JavaScript SDK*
- *Glossary*
- Content of the *Docker Configuration* section has been transferred into the new section *Preparing to work*
- The section Docker smart contracts with the use of the node REST API has been deleted from the index

29.3 1.4.0

The following pages have been added:

- *Atomic transactions*
- *Working in the web client*
- *JavaScript SDK*

The following sections have been changed:

- *Architecture*
- *Transactions*
- *Authorization type configuration for the REST API and gRPC access*
- Node API tools
- *Node installation*

29.4 1.3.1

The following pages have been added:

- *Parallel contract execution*

The following sections have been changed:

- *Creating a smart contract*
- *Docker configuration*

29.5 1.3.0

The following sections have been changed:

- *Client*
- The "Role model" and "Access managing" sections have been converted to a section *Permissions managing*
- *Description of the node configuration file parameters and sections*
- *Privacy data access groups configuration*
- *Docker configuration*
- *Addresses* REST API methods
- *Node* REST API methods
- *Contracts* REST API methods
- *Privacy* REST API methods
- *System requirements*

29.6 1.2.3

The following sections have been changed:

- *Docker Smart Contracts*
- *Description of the node configuration file parameters and sections*
- *Privacy data access groups configuration*

29.7 1.2.2

The following pages have been added:

- REST API *Debug* methods
- Full REST API description on the API Docs page

The following sections have been changed:

- *Installing and running the Waves Enterprise platform*

29.8 1.2.0

The following pages have been added:

- A new section *Integration services*, which includes *Authorization service* and *Data preparation service*
- *Obtaining a license* section was added
- A new REST API *Licenses* method was added
- A new *Smart contract run with gRPC* section was added
- A new *gRPC services available to smart contract* section was added

The following sections have been changed:

- *Installing and running the Waves Enterprise platform*
- The *Cryptography* section was renovated. Part of information was moved into *Data encryption operations* section
- *Changes in the node configuration file*
- *Transactions*

29.9 1.1.2

The following sections have been changed:

- Sandbox
- *Changes in the node configuration file*
- *Node installation* was converted into "Installing and running the Waves Enterprise platform"
- *Participants connection to the network*

- *Anchoring settings*
- *Authorization type configuration for the REST API access*
- *Connection of the node to the "Waves Enterprise Partnernet"*
- *Connection of the node to the "Waves Enterprise Mainnet"*
- *System requirements*

29.10 1.1.0

The following pages have been added:

- *API methods available to smart contract*
- Sandbox
- *Changes in the node configuration file*

The following sections have been changed:

- Docker Smart Contracts
- Example of starting a contract
- *Node installation*
- Additional services deploy

29.11 1.0.0

The following pages have been added:

- *Authorization service*

The following sections have been rebuilt:

- *Node configuration*
- *Mainnet and Partnernet connection*
- REST API
- *Node installation*

Changes in the node configuration file `node.conf`

- The *NTP server* section is added
- The `auth` section is added into the authorization type selection of the *REST API* section