Whiteblock User Guide Documentation

Release 1.0

Trey Zhong

Dec 22, 2018

Contents

Welcome to the Whiteblock User Guide.

CHAPTER 1

Overview

Note: Please note that these documents are still being drafted and subject to change.

Whiteblock is a full-stack blockchain testing platform that helps development teams quickly test & deploy high-performing distributed ledger technologies.

The Whiteblock platform allows users to provision multiple fully-functioning nodes (>1000) within a private test network over which they have complete control. Transactions, behaviors, and other logic can be automated as nodes interact in real-time, allowing testers to observe the performance of protocols, algorithms, decentralized applications, or changes to system infrastructure prior to deployment.

Each node exists within its own Virtual Local Area Network (VLAN) and is assigned a unique IP address. This provides logical separation between nodes. The links between these nodes can be configured with latency, packet loss, and other WAN conditions according to user specification in order to replicate real-world performance that is accurate within .01 milliseconds.

1.1 Use Cases

- Simulate Live, Global Network Performance
- Configure Latency, Packet Loss, & Bandwidth.
- Real-Time Data & Statistics
- Provision & Configure Multiple Independent Nodes
- Test Fault Tolerance
- Automate Transactions By Size, Frequency, & Rate.
- Private, Integrated Development Environment
- Compile, Deploy, & Functionally Test Smart Contracts

1.2 Features

- **Blockchain Agnostic** - Deploy from list of supported clients with appropriate testing libraries or bring your own into the Whiteblock environment.
- **Automate Genesis Ceremony** - Quickly provision a functional blockchain network without the additional hassle of editing a configuration file every time a new test network is provisioned.
- **Account & Wallet Creation** - Indicate total number of accounts/wallets to be created within network.
- **Pre-allocate Funds** - Fund wallets and accounts with a specified amount of assets. Start testing right away without the need to wait for mining rewards.
- **Transactional Logic** - Indicate the value and volume of transactions to automate activity. Measure transactions per second (TPS) and observe the effects of various environmental conditions on throughput.
- **Network Behavior** - Configure Wide Area Network (WAN) conditions between nodes, including latency, packet loss, and bandwidth constraints to replicate real-world performance within a safe and controlled environment.
- **Data Visualization** - The Whiteblock Explorer provides insight into TPS, uncle rate, and other relevant data points and performance metrics.
- **Reporting** - Generate concise reports to summarize your test cases.
- **Pipeline Integration** - Whiteblock seamlessly integrates with your CI/CD platform of choice to reduce development overhead.

1.3 Network Specifications

- Bandwith Capabilities: Up To 1 00Gb/s
- Maximum Nodes: >20,000
- Maximum Latency: 20S I 20,000 MS
- Packet Loss: Random, Periodic, Burst, BER, Gilbert-Elliot Time Accuracy: Within .01 Milliseconds

1.4 Supported Clients

While the platform itself is blockchain agnostic, several clients are natively supported.

Some of these clients include:

- Ethereum (Geth & Parity)
- Bitcoin
- Syscoin
- EOS
- Hyperledger Fabric
- Hyperledger Sawtooth
- Quorum
- NEM
- Monero

Note: Do you have a client of your own that you would like Whiteblock to support? Feel free to contact the team via email at *hello@whiteblock.io* or join our Telegram channel

CHAPTER 2

Get Started

There are two primary components of the Whiteblock platform:

Graphical User Interface (GUI) - The GUI acts as a data visualization dashboard which provides insight into relevant performance metrics for the blockchain network.

Command Line Interface (CLI) - Using the CLI application, users can interface with the Whiteblock platform. It provides functionality which allows users to configure the blockchain network, provision nodes, and automate various behaviors and actions within the blockchain network. The following documentation focuses primarily on the CLI application.

2.1 Accessing The GUI

New users will be provided a custom subdomain on the Whiteblock website. You will be provided user credentials when you sign up. Simply use the specified username and password to log-in to the GUI.

2.2 Accessing The CLI Application

The Whiteblock team will provide a unique public/private keypair that will allow you to securely connect to the CLI application within a private Whiteblock environment through a Secure Shell (SSH) session.

2.3 Build The Network

In this step, we create and deploy a selected blockchain network with the specified client and number of nodes. The Whiteblock platform provisions each node as an independent participant within the blockchain network. Nodes are logically separated from one another within their own Local Area Network (LAN).

After running the build command as shown below, you will be prompted to customize the blockchain network, this includes configuring the Wide Area Network (WAN) links between each node with latency, packet loss, bandwidth

constraints, and other common network characteristics. If this is your first time using the Whiteblock platform, it may be easier to stick with the default parameters provided for each option.

Command:

```
$ whiteblock build
```

Output:

```
blockchain (default set to: ethereum):
nodes (default set to: 10):
image (default set to: ethereum:latest):
cpus (default set to: no limit):
memory (default set to: no limit):
Use default parameters? (y/n) y
2018/12/13 20:07:50 build:  Build in Progress
Building: 100.000000
Done
```

2.4 Configure Network Conditions

To add network latency between nodes, use the following commands:

```
$ whiteblock netconfig delay 1 1 200
```

The above commands configure the amount of latency between nodes and follows this format: delay <engine number> <path number> <amount>.

The first *1* value refers to emulation engine 1, the second 1 refers to VLAN path 1. The *200* valuerefers to the total amount of one-way latency, which translates to milliseconds.

```
$ whiteblock netconfig on 1
```

This command turns the netconfig engine on and implements latency on on engine 1.

To turn disbale netconfig, use the following command:

```
$ whiteblock netconfig off 1
```

For more advanced netconfig parameters, please visit the Command Line Reference page.

2.5 Automate Transactions

After configuring network conditions, transactional logic can be defined and automated for such purposes as throughput tests. Transaction commands adhere to the following format: whiteblock <blockchain-interface> send_transactions <tx/s> <value>. The <blockchain-interface> needs to be consistent with the relevant command used by the client that was indicated when the network was built.

The transaction engine will automate transactions according to the specified submission rate in the second argument <tx/s> and the amount of assets sent in the third parameter <value>, which is specified in hex. This will immediately begin transactions once the network is finished building, but these values can also be configured and altered once the network has already been built.

To start transactions, run the following command:

```
$ whiteblock geth start_transactions 100 0x545454
started
```

To stop the transaction, run the following command

```
$ whiteblock geth stop_transactions
success
```

Note: currently we only support geth for sending transaction through command line. To send transaction for other type of blockchains, you can use Websocket API calls. Please refer to the Generics section in the Websocket API in *References* for more information.

2.6 Examine Data

You may now go to the GUI and use our data visualization tools to examine the different data points that are being push directly from the blockchain.

If you want to quickly check the stats of your current blockchain network, use the following command.

Command:

```
$ whiteblock get stats all
```

Output:


```
{
  "blockTime": {
    "max": 70,
    "mean": 1.2978947368421072,
    "standardDeviation": 1.7608896643379766
  },
  "difficulty": {
    "max": 329333,
    "mean": 214993.2977380325,
    "standardDeviation": 56914.20143516361
  },
  "gasLimit": {
    "max": 8000000,
    "mean": 7168060.679642294,
    "standardDeviation": 1286432.4077131029
  },
  "gasUsed": {
    "max": 7917000,
    "mean": 534323.5139400318,
    "standardDeviation": 1538475.9696957779
  },
  "totalDifficulty": {
    "max": 408802259,
    "mean": 173546242.58337703,
    "standardDeviation": 117177703.83311588
  },
  "tps": {
    "max": 377,
    "mean": 18.855407894736842,
    "standardDeviation": 58.25808243503218
```


(continues on next page)

(continued from previous page)


```
  },
  "transactionCount": {
    "max": 377,
    "mean": 25.443976854287218,
    "standardDeviation": 73.26076046170377
  },
  "uncleCount": {
    "max": 1,
    "mean": 0.11204629142556508,
    "standardDeviation": 0.3154233979959995
  }
}
```


To learn more about how to use our command line features, please visit the *References* page.

CHAPTER 3

Command Line Interface

```
$ whiteblock <command> [FLAGS]
```

This application will deploy a blockchain, create nodes, and allow those nodes to interact in the network.

3.1 Available Commands

- **Available Commands:**
  - *build* Build a blockchain using image and deploy nodes
  - *get* Get server and network information.
  - *geth* Run geth commands
  - *help* Displays help page
  - *netconfig* Network conditions
  - *rpc* Rpc interacts with the blockchain
  - *ssh* SSH into an existing container.
  - *version* Display Whiteblock CLI version
- **Flags:**
  - -h, –help : help for whiteblock

3.2 Build

```
whiteblock build [FLAGS]
```

Aliases: build, create, init

Build will create and deploy a blockchain and the specified number of nodes. Each node will be instantiated in its own container and will interact individually as a participant of the specified network.

- **Flags:**
  - -h, –help: help for build
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

3.2.1 Ethereum

3.2.1.1 Build

```
$ whiteblock build
```

This application will deploy an Ethereum blockchain, create nodes, and allow those nodes to interact in the network.

3.2.1.2 Mining

For Ethereum, the command to start mining is given below. If no arguments are given, this command will have all nodes start mining. Users can specify the number of nodes to mine by giving the node numbers as arguments to this command.

```
$ whiteblock geth start_mining
```

3.2.1.3 Automate Transactions

After configuring network conditions, transactional logic can be defined and automated for such purposes as throughput tests.

In the case for Ethereum, this command will automate transactions according to the specified submission rate in the second argument <tx/s> and the amount of assets sent in the third parameter <value>, which is specified in hex. A destination can be specified, but if none is given, all nodes will send transactions to the next node (round robin).

This will immediately begin transactions once the network is finished building, but these values can also be configured and altered once the network has already been built.

To start transactions, run the following command:

```
$ whiteblock geth start_transactions 100 0x545454
```

To stop the transaction, run the following command:

```
$ whiteblock geth stop_transactions
```

To send transaction for other type of blockchains, the user can use Websocket API calls. Please refer to the Generics section in the Websocket API in *References* for more information.

3.2.2 Rchain

3.2.2.1 Build

```
$ whiteblock build
```

Build Using Default Parameters:

```
blockchain (default set to: rchain):
nodes (default set to: 10):
image (default set to: rchain:latest):
cpus (default set to: no limit):
memory (default set to: no limit):
Use default parameters? (y/n) y
2018/12/19 23:24:12 build:  Build in Progress
Building: 100.000000
Done
```

To Build Without Default Parameters, Type n When Being Prompted: *Use default parameters? (y/n)*

```
blockchain (default set to: rchain):
nodes (default set to: 10): 15
image (default set to: rchain:latest):
cpus (default set to: no limit):
memory (default set to: no limit):
Use default parameters? (y/n) n
chainId (int):
networkId (int):
difficulty (int):
initBalance (string):
maxPeers (int):
gasLimit (int):
homesteadBlock (int):
eip155Block (int):
eip158Block (int):
2018/12/19 23:24:12 build:  Build in Progress
Building: 100.000000
Done
```

3.2.3 Syscoin

3.2.3.1 Build

<Blank>

3.3 Data Retrieval

3.3.1 Ethereum Geth Commands

```
whiteblock geth <command> [FLAGS]
```

Geth will allow the user to get information and run geth commands.

- **Available SubCommands:**
  - block_listener Get block listener
  - get_accounts Get account information
  - get_balance Get account balance information
  - get_block Get block information
  - get_block_number Get block number
  - get_hash_rate Get hasg rate
  - get_recent_sent_tx Get recently sent transaction
  - get_transaction Get transaction information
  - get_transaction_count Get transaction count
  - get_transaction_receipt Get transaction receipt
  - send_transaction Sends a transaction
  - start_mining Start Mining
  - start_transactions Start transactions
  - stop_mining Stop mining
  - stop_transactions Stop transactions
- **Flags:**
  - -h, –help: help for geth
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

geth block_listener

```
whiteblock geth block_listener [block number] [FLAGS]
```

Get all blocks and continue to subscribe to new blocks

Format: [block number] Params: The block number to start at or None for all blocks Response: Will emit on eth::block_listener for every block after the given block or 0 that exists/has been created

- **Flags:**
  - -h, –help: help for block_listener

geth get_accounts

```
whiteblock geth get_accounts [FLAGS]
```

Get a list of all unlocked accounts

Response: A JSON array of the accounts

- **Flags:**
  - -h, –help: help for get_accounts

geth get_balance

```
whiteblock geth get_balance <address> [FLAGS]
```

Get the current balance of an account

Format: <address> Params: Account address Response: The integer balance of the account in wei

- Flags: - -h, –help: help for get_balance

geth get_block

```
whiteblock geth get_block <block number> [FLAGS]
```

Get the data of a block

Format: <Block Number> Params: Block number

- **Flags:**
  - -h, –help: help for get_block

geth get_block_number

```
whiteblock geth get_block_number [FLAGS]
```

Get the current highest block number of the chain

Response: The block number

- **Flags:**
  - -h, –help: help for get_block_number

geth get_hash_rate

```
whiteblock geth get_hash_rate [FLAGS]
```

Get the current hash rate per node

Response: The hash rate of a single node in the network

- **Flags:**
  - -h, –help: help for get_hash_rate

geth get_recent_sent_tx

```
whiteblock geth get_recent_sent_tx [NUMBER] [FLAGS]
```

Get a number of the most recent transactions sent

Format: [number] Params: The number of transactions to retrieve Response: JSON object of transaction data

- **Flags:**
  - -h, –help: help for get_recent_sent_tx

geth get_transaction

```
whiteblock geth get_transaction <hash> [FLAGS]
```

Get a transaction by its hash

Format: <hash> Params: The transaction hash

Response: JSON representation of the transaction.

- **Flags:**
  - -h, –help: help for get_transaction

geth get_transaction_count

```
whiteblock geth get_transaction_count <address> [BLOCK NUMBER] [FLAGS]
```

Get the transaction count sent from an address, optionally by block

Format: <address> [block number] Params: The sender account, a block number Response: The transaction count

- **Flags:**
  - -h, –help: help for get_transaction_count

geth get_transaction_receipt

```
whiteblock geth get_transaction_receipt <hash> [FLAGS]
```

Get the transaction receipt by the tx hash

Format: <hash> Params: The transaction hash Response: JSON representation of the transaction receipt.

- **Flags:**
  - -h, –help: help for get_transaction_receipt

geth send_transaction

```
whiteblock geth send_transaction <from address> <to address> <gas> <gas price> <value␣
↪to send> [FLAGS]
```

Send a transaction between two accounts

Format: <from> <to> <gas> <gas price> <value> Params: Sending account, receiving account, gas, gas price, amount to send, transaction data, nonce Response: The transaction hash

- Flags: - -h, –help: help for send_transaction

geth start_mining

```
whiteblock geth start_mining [node 1 number] [node 2 number]... [FLAGS]
```

Send the start mining signal to nodes, may take a while to take effect due to DAG generation

Format: [node 1 number] [node 2 number]... Params: A list of the nodes to start mining or None for all nodes Response: The number of nodes which successfully received the signal to start mining

- **Flags:**
  - -h, –help: help for start_mining

geth start_transactions

```
whiteblock geth start_transactions <tx/s> <value> [DESTINATION] [FLAGS]
```

Start sending transactions according to the given parameters, value = -1 means randomize value.

Format: <tx/s> <value> [destination] Params: The amount of transactions to send in a second, the value of each transaction in wei, the destination for the transaction

- **Flags:**
  - -h, –help: help for start_transactions
  - geth stop_mining

geth stop_mining

```
whiteblock geth stop_mining [node 1 number] [node 2 number]... [FLAGS]
```

Send the stop mining signal to nodes

Format: [node 1 number] [node 2 number]... Params: A list of the nodes to stop mining or None for all nodes Response: The number of nodes which successfully received the signal to stop mining

- **Flags:**
  - -h, –help: help for stop_mining

geth stop_transactions

```
whiteblock geth stop_transactions [FLAGS]
```

Stops the sending of transactions if transactions are currently being sent

- **Flags:**
  - -h, –help: help for stop_transactions

Geth (Go-Ethereum)

Note: Any configuration option can be left out, and this entire section can even be null, the example contains all of the defaults

Options

- chainId: The chain id set in the genesis.conf
- networkId: The network id
- difficulty: The initial difficulty set in the genesis.conf file
- initBalance: The initial balance for the accounts
- maxPeers: The maximum number of peers for each node
- gasLimit: The initial gas limit
- homesteadBlock: Set in genesis.conf
- eip155Block: Set in genesis.conf
- eip158Block: Set in genesis.conf

Example (using defaults)

```
{
    "chainId":15468,
    "networkId":15468,
    "difficulty":100000,
    "initBalance":100000000000000000000,
    "maxPeers":1000,
    "gasLimit":4000000,
    "homesteadBlock":0,
    "eip155Block":0,
    "eip158Block":0
}
```

3.3.2 Rchain

<Blank>

3.3.3 Syscoin Sys Commands

Options:

- rpcUser: The username credential
- rpcPass: The password credential
- masterNodeConns: The number of connections to set up for the master nodes
- nodeConns: The number of connections to set up for the normal nodes
- percentMasternodes: The percentage of the network consisting of master nodes
- options: Options to set enabled for all nodes
- senderOptions: Options to set enabled for senders
- receiverOptions: Options to set enabled for receivers
- mnOptions: Options to set enabled for master nodes
- extras: Extra options to add to the config file for all nodes
- senderExtras: Extra options to add to the config file for senders
- receiverExtras: Extra options to add to the config file for receivers
- mnExtras: Extra options to add to the config file for master nodes

```
whiteblock sys <command> [FLAGS]
```

Alias: SYS, syscoin

Sys will allow the user to get information and run SYS commands.

- **Available Commands:**
  - test SYS test commands.
- **Flags:**
  - -h, –help : help for sys

sys test

```
whiteblock sys test <command> [FLAGS]
```

Available Commands: results Get results from a previous test. start Starts propagation test.

- **Flags:**
  - -h, –help : help for test

sys test start

```
whiteblock sys test start <wait time> <min complete percent> <number of tx> [FLAGS]
```

Sys test start will start the propagation test. It will wait for the signal start time, have nodes send messages at the same time, and require to wait a minimum amount of time then check receivers with a completion rate of minimum completion percentage.

Format: <wait time> <min complete percent> <number of tx> Params: Time in seconds, percentage, number of transactions

- **Flags:**

  - -h, –help : help for start
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

sys test results

```
whiteblock sys test results <test number> [FLAGS]
```

Sys test results pulls data from a previous test or tests and outputs as csv.

Format: <test number> Params: Test number

- **Flags:**
  - -h, –help : help for results
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

3.4 Get Commands

```
whiteblock get <command> [FLAGS]
```

Get will ouput server and network information and statstics.

- **Available Commands:**
  - data Data will pull data from the network and output into a file.
  - nodes Nodes will show all nodes in the network.
  - server Get server information.
  - stats Get stastics of a blockchain
- **Flags:**
  - -h, –help : help for get
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get data

```
whiteblock get data <command> [FLAGS]
```

Data will pull specific or all block data from the network and output into a file. You will specify the directory where the file will be downloaded.

- **Available Commands:**
  - all All will pull data from the network and output into a file.
  - block Data block will pull data from the network and output into a file.
  - time Data time will pull data from the network and output into a file.
- **Flags:**
  - -h, –help : help for data
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get data all

```
whiteblock get data all [PATH] [FLAGS]
```

Data all will pull all data from the network and output into a file. The directory where the file will be downloaded will need to be specified. If no directory is provided, default directory is set to ~/Downloads.

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for all
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get data block

```
whiteblock get data block <start block> <end block> [PATH] [FLAGS]
```

Data block will pull block data from the network from a given start and end block and output into a file. The directory where the file will be downloaded will need to be specified. If no directory is provided, default directory is set to ~/Downloads.

Params: Block numbers Format: <start block number> <end block number>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for block
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get data time

```
whiteblock get data time <start time> <end time> [PATH] [FLAGS]
```

Data time will pull block data from the network from a given start and end time and output into a file. The directory where the file will be downloaded will need to be specified. If no directory is provided, default directory is set to ~/Downloads.

Params: Unix time stamps Format: <start unix time stamp> <end unix time stamp>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for time
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get nodes

```
whiteblock get nodes [FLAGS]
```

Aliases: nodes, node

Nodes will output all of the nodes in the current network.

- **Flags:**
  - -h, –help : help for server
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get server

```
whiteblock get server [FLAGS]
```

Aliases: server, servers

Server will allow the user to get server information.

- **Flags:**
  - -h, –help : help for server
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats

```
whiteblock get stats <command> [FLAGS]
```

Stats will allow the user to get statistics regarding the network.

Response: JSON representation of network statistics

- **Available Commands:**
  - all
  - block
  - time
- **Flags:**
  - -h, –help : help for stats
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats all

```
whiteblock get stats all [FLAGS]
```

Stats all will allow the user to get all the statistics regarding the network.

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for all
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats block

```
whiteblock get stats block <start block> <end block> [FLAGS]
```

Stats block will allow the user to get statistics regarding the network.

Params: Block numbers Format: <start block number> <end block number>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for block
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats time

```
whiteblock get stats time <start time> <end time> [FLAGS]
```

Stats time will allow the user to get statistics by specifying a start time and stop time (unix time stamp).

Params: Unix time stamps Format: <start unix time stamp> <end unix time stamp>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for time
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

3.5 Netconfig

```
whiteblock netconfig <command> [FLAGS]
```

Netconfig will introduce persistnace network conditions for testing. Use '?' at any time for more help on configuring the network.

Custom Command: netconfig <engine number> <path number> <command>

set delay <amount> Specifies the latency to add [ms]; set loss loss <amount> Specifies the amount of packet loss to add [%]; set bw <amount> <type> Specifies the bandwidth of the network [bps|Kbps|Mbps|Gbps];

- **Available Commands:**
  - bandwidth Set bandwidth
  - delay Set latency
  - loss Set packetloss
  - off Turn off emulation
  - on Turn on emulation
- **Flags:** -h, –help: help for netconfig

netconfig bandwidth

```
whiteblock netconfig bandwidth <engine number> <path number> <amount> <bandwidth type>
↪ [FLAGS]
```

Aliases: bw

Bandwidth will constrict the network to the specified bandwidth. You will specify the amount of bandwdth and the type.

Fomat: bandwidth type: bps, Kbps, Mbps, Gbps

- **Flags:**
  - -h, –help: help for bandwidth

netconfig delay

```
whiteblock netconfig delay <engine number> <path number> <amount> [FLAGS]
```

Aliases: delay, latancy, lat

Latency will introduce delay to the network. You will specify the amount of latency in ms.

- **Flags:**
  - -h, –help: help for latency

netconfig loss

```
whiteblock netconfig loss <engine number> <path number> <percent> [FLAGS]
```

Aliases: packetloss

Packetloss will drop packets in the network. You will specify the amount of packet loss in %.

- **Flags:**
  - -h, –help: help for loss

netconfig off

```
whiteblock netconfig off <engine number> [FLAGS]
```

Turn off emulation.

- **Flags:**
  - -h, –help: help for off

netconfig on

```
whiteblock netconfig on <engine number> [FLAGS]
```

Turn on emulation.

- **Flags:**
  - -h, –help: help for on

3.6 SSH

```
$ whiteblock ssh <server> <node> [FLAGS]
```

SSH will allow the user to go into the contianer where the specified node exists.

Response: stdout of the command

- **Flags:**
  - -h, –help : help for ssh
  - -a, –server-addr : server address with port 5000 (default “localhost:5000”)

3.7 Version

```
$ whiteblock version
```

Get whiteblock CLI client version

Flags: -h, –help : help for version

<Intentionally left blank>

CHAPTER 4

Client-specific Command Lines

4.1 Ethereum

4.1.1 Build

4.1.2 Mining

4.1.3 Automate Transactions

4.1.4 Geth

4.2 Syscoin

4.2.1 Build

4.2.2 Sys

4.3 Rchain

4.3.1 Build

CHAPTER 5

References

Note: Please note that these documents are still being drafted and subject to change. This page is particularly rough, so take it easy, buddy.

5.1 Command Line Interface

```
whiteblock <command> [FLAGS]
```

This application will deploy a blockchain, create nodes, and allow those nodes to interact in the network.

- **Available Commands:**
  - *build* Build a blockchain using image and deploy nodes
  - *get* Get server and network information.
  - *geth* Run geth commands
  - *help* Displays help page
  - *netconfig* Network conditions
  - *rpc* Rpc interacts with the blockchain
  - *ssh* SSH into an existing container.
  - *version* Display Whiteblock CLI version
- **Flags:**
  - -h, –help : help for whiteblock

5.2 Fowarding Commands

- **forward**

  - forward {“nodes”:[<node1>,...<noden>],”port”:<port>,”data”:”<data to send to all the nodes given>”}
  - forward tcp data to specific nodes, pass data with c string escapes and receive data back with c string escapes, in a json array of strings
  - Example: {“nodes”:[1,2],”port”:80,”data”:”GET / HTTP/1.1rnrn”}
  - Response: JSON Array of the responses
  - Response Example: [“HTTP/1.1 200 OKrnrn”]

5.3 build

```
whiteblock build [FLAGS]
```

Aliases: build, create, init

Build will create and deploy a blockchain and the specified number of nodes. Each node will be instantiated in its own container and will interact individually as a participant of the specified network.

- **Flags:**
  - -h, –help: help for build
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

5.4 get

```
whiteblock get <command> [FLAGS]
```

Get will ouput server and network information and statstics.

- **Available Commands:**
  - data Data will pull data from the network and output into a file.
  - nodes Nodes will show all nodes in the network.
  - server Get server information.
  - stats Get stastics of a blockchain
- **Flags:**
  - -h, –help : help for get
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get data

```
whiteblock get data <command> [FLAGS]
```

Data will pull specific or all block data from the network and output into a file. You will specify the directory where the file will be downloaded.

- **Available Commands:**
  - all All will pull data from the network and output into a file.
  - block Data block will pull data from the network and output into a file.

  - time Data time will pull data from the network and output into a file.

- **Flags:**
  - -h, –help : help for data
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get data all

```
whiteblock get data all [PATH] [FLAGS]
```

Data all will pull all data from the network and output into a file. The directory where the file will be downloaded will need to be specified. If no directory is provided, default directory is set to ~/Downloads.

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for all
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get data block

```
whiteblock get data block <start block> <end block> [PATH] [FLAGS]
```

Data block will pull block data from the network from a given start and end block and output into a file. The directory where the file will be downloaded will need to be specified. If no directory is provided, default directory is set to ~/Downloads.

Params: Block numbers Format: <start block number> <end block number>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for block
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get data time

```
whiteblock get data time <start time> <end time> [PATH] [FLAGS]
```

Data time will pull block data from the network from a given start and end time and output into a file. The directory where the file will be downloaded will need to be specified. If no directory is provided, default directory is set to ~/Downloads.

Params: Unix time stamps Format: <start unix time stamp> <end unix time stamp>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for time
  - -a, –server-addr string: server address with port 5000 (default "localhost:5000")

get nodes

```
whiteblock get nodes [FLAGS]
```

Aliases: nodes, node

Nodes will output all of the nodes in the current network.

- **Flags:**
  - -h, –help : help for server
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get server

```
whiteblock get server [FLAGS]
```

Aliases: server, servers

Server will allow the user to get server information.

- **Flags:**
  - -h, –help : help for server
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats

```
whiteblock get stats <command> [FLAGS]
```

Stats will allow the user to get statistics regarding the network.

Response: JSON representation of network statistics

- **Available Commands:**
  - all
  - block
  - time
- **Flags:**
  - -h, –help : help for stats
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats all

```
whiteblock get stats all [FLAGS]
```

Stats all will allow the user to get all the statistics regarding the network.

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for all
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats block

```
whiteblock get stats block <start block> <end block> [FLAGS]
```

Stats block will allow the user to get statistics regarding the network.

Params: Block numbers Format: <start block number> <end block number>

Response: JSON representation of network statistics

- **Flags:**

  - -h, –help : help for block
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

get stats time

```
whiteblock get stats time <start time> <end time> [FLAGS]
```

Stats time will allow the user to get statistics by specifying a start time and stop time (unix time stamp).

Params: Unix time stamps Format: <start unix time stamp> <end unix time stamp>

Response: JSON representation of network statistics

- **Flags:**
  - -h, –help : help for time
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

5.5 netconfig

```
whiteblock netconfig <command> [FLAGS]
```

Netconfig will introduce persistnace network conditions for testing. Use ‘?’ at any time for more help on configuring the network.

Custom Command: netconfig <engine number> <path number> <command>

set delay <amount> Specifies the latency to add [ms]; set loss loss <amount> Specifies the amount of packet loss to add [%]; set bw <amount> <type> Specifies the bandwidth of the network [bps|Kbps|Mbps|Gbps];

- **Available Commands:**
  - bandwidth Set bandwidth
  - delay Set latency
  - loss Set packetloss
  - off Turn off emulation
  - on Turn on emulation
- **Flags:** -h, –help: help for netconfig

netconfig bandwidth

```
whiteblock netconfig bandwidth <engine number> <path number> <amount> <bandwidth type>
↪ [FLAGS]
```

Aliases: bw

Bandwidth will constrict the network to the specified bandwidth. You will specify the amount of bandwdth and the type.

Fomat: bandwidth type: bps, Kbps, Mbps, Gbps

- **Flags:**
  - -h, –help: help for bandwidth

netconfig delay

```
whiteblock netconfig delay <engine number> <path number> <amount> [FLAGS]
```

Aliases: delay, latancy, lat

Latency will introduce delay to the network. You will specify the amount of latency in ms.

- **Flags:**
  - -h, –help: help for latency

netconfig loss

```
whiteblock netconfig loss <engine number> <path number> <percent> [FLAGS]
```

Aliases: packetloss

Packetloss will drop packets in the network. You will specify the amount of packet loss in %.

- **Flags:**
  - -h, –help: help for loss

netconfig off

```
whiteblock netconfig off <engine number> [FLAGS]
```

Turn off emulation.

- **Flags:**
  - -h, –help: help for off

netconfig on

```
whiteblock netconfig on <engine number> [FLAGS]
```

Turn on emulation.

- **Flags:**
  - -h, –help: help for on

5.6 SSH

```
whiteblock ssh <server> <node> [FLAGS]
```

SSH will allow the user to go into the contianer where the specified node exists.

Response: stdout of the command

- **Flags:**
  - -h, –help : help for ssh
  - -a, –server-addr : server address with port 5000 (default “localhost:5000”)

5.7 version

```
whiteblock version
```

Get whiteblock CLI client version

- Flags: - -h, –help : help for version

5.8 Smart Contracts

contractadd

```
whiteblock contractadd <filename> [FLAGS]
```

Adds the specified smart contract into the /Downloads folder.

- **Flags:**
  - -h, –help: help for contractadd
  - -p, –path string : File path where the smart contract is located

contractcompile

```
whiteblock contractcompile <filename> [FLAGS]
```

Compiles the specified smart contract.

- **Flags:**
  - -h, –help: help for contractcompile
  - -p, –path string: File path where the smart contract is located

5.9 Ethereum

- **eth::get_block_number**
  - Description: Get the current highest block number of the chain
  - Params: None
  - Response: The block number e.g. 10
- **eth::get_block**
  - Description: Get the data of a block
  - Params: The block number
  - Format: <Block Number>
  - Example: 10
  - Response: JSON Representation of the block. Example
- **eth::get_accounts**
  - Description: Get the unlocked accounts

  - Params: None
  - Response: A JSON array of the accounts
- **eth::get_balance**
  - Description: Get the current balance of an account
  - Params: Account address
  - Format: <address>
  - Example: 0xbfa767eae64753e4c426ea42470abf7e4fc305ab
  - Response: The integer balance of the account in wei
- **eth::send_transaction**
  - Description: Send a transaction between two accounts
  - Params: Sending account, receiving account, gas, gas price, amount to send, transaction data, nonce
  - Format: <from> <to> <gas> <gas price> <value> [data] [nonce]
  - Example: 0xbfa767eae64753e4c426ea42470abf7e4fc305ab 0x8d12a197cb00d4747a1fe03395095ce2a5cc6819 0x015f90 0x165a0bc00 0xde0b6b3a7640000
  - Response: The transaction hash
- **eth::get_transaction_count**
  - Description: Get the transaction count sent from an address, optionally by block
  - Params: The sender account, a block number
  - Format: <address> [block number]
  - Example: 0xbfa767eae64753e4c426ea42470abf7e4fc305ab
  - Response: The transaction count
- **eth::get_transaction**
  - Description: Get a transaction by its hash
  - Params: The transaction hash
  - Format: <hash>
  - Example: 0x402c257c85c398154b8b16fa612df13e197135f63d1be9e03b6d2d55285e8670
  - Response: JSON representation of the transaction. Example
- **eth::get_transaction_receipt**
  - Description: Get the transaction receipt by the tx hash
  - Params: The transaction hash
  - Format: <hash>
  - Example: 0x402c257c85c398154b8b16fa612df13e197135f63d1be9e03b6d2d55285e8670
  - Response: JSON representation of the transaction receipt. Example
- **eth::get_hash_rate**
  - Description: Get the current hash rate per node
  - Params: None

  - Response: The hash rate of a single node in the network

- **eth::start_transactions**
  - Description: Start sending transactions according to the given parameters, value = -1 means randomize value.
  - Params: The amount of transactions to send in a second, the value of each transaction in wei, the destination for the transaction
  - Format: <tx/s> <value> [destination]
  - Example: 100 0xde0b6b3a7640000 0x8d12a197cb00d4747a1fe03395095ce2a5cc6819
  - Response: None
- **eth::stop_transactions**
  - Description: Stops the sending of transactions if transactions are currently being sent
  - Params: None
  - Response: None
- **eth::start_mining**
  - Description: Send the start mining signal to nodes, may take a while to take effect due to DAG generation
  - Params: A list of the nodes to start mining or None for all nodes
  - Format: [node 1 number] [node 2 number]...
  - Example: 0 1 2 3
  - Response: The number of nodes which successfully received the signal to start mining
- **eth::stop_mining**
  - Description: Send the stop mining signal to nodes
  - Params: A list of the nodes to stop mining or None for all nodes
  - Format: [node 1 number] [node 2 number]...
  - Example: 0 1 2 3
  - Response: The number of nodes which successfully received the signal to stop mining
- **eth::block_listener**
  - Description: Get all blocks and continue to subscribe to new blocks
  - Params: The block number to start at or None for all blocks
  - Format: [block number]
  - Example: 12
  - Response: Will emit on eth::block_listener for every block after the given block or 0 that exists/has been created
- **eth::get_recent_sent_tx**
  - Description: Get a number of the most recent transactions sent
  - Params: The number of transactions to retrieve
  - Format: [number]

- Example: 5
- Response: Data on the 5 last sent transactions
- Response Example:


```
{"results":[{"statement_id":0,"series":[{"name":"transactions","columns":[
↪"time","from","gas","gas_price","to","txid","value"],"values":[["2018-11-
↪08T18:02:59.700086831Z","\"0x1949d6d0dfb19048563b602d9a02c06420421429\"","\
↪"0x15f90\"","\"0x3B9ACA00\"","\"0xd9075634d9725f05a1a84343fb40a31d9964ffa5\"
↪","\"0xaffad4a457d79448f211654be8eae1ca6fa8e005936d72528d394fe724adb903\"",
↪"0xDE0B6B3A7640000"],["2018-11-08T18:02:59.698273467Z","\
↪"0x1949d6d0dfb19048563b602d9a02c06420421429\"","\"0x15f90\"","\"0x3B9ACA00\"
↪","\"0xd9075634d9725f05a1a84343fb40a31d9964ffa5\"","\
↪"0x8f08bc904c7fbf2e3c695bd71237432137e4f22a20287eda880ed8b409032580\"",
↪"0xDE0B6B3A7640000"],["2018-11-08T18:02:59.655393436Z","\
↪"0xd9075634d9725f05a1a84343fb40a31d9964ffa5\"","\"0x15f90\"","\"0x3B9ACA00\"
↪","\"0xe33e509fea81ea03333a3659c98108196ac438a7\"","\
↪"0x21ed0c41959ec9aecf36461cd5b42e65505090e8dbd514ba3b123a3889a5735e\"",
↪"0xDE0B6B3A7640000"],["2018-11-08T18:02:59.651551261Z","\
↪"0x1949d6d0dfb19048563b602d9a02c06420421429\"","\"0x15f90\"","\"0x3B9ACA00\"
↪","\"0xd9075634d9725f05a1a84343fb40a31d9964ffa5\"","\
↪"0xfc9b2658bdc95669ffd38e8ff02b9995d894542db52161fbe41ee5dcaed70628\"",
↪"0xDE0B6B3A7640000"],["2018-11-08T18:02:59.628233357Z","\
↪"0xd9075634d9725f05a1a84343fb40a31d9964ffa5\"","\"0x15f90\"","\"0x3B9ACA00\"
↪","\"0xe33e509fea81ea03333a3659c98108196ac438a7\"","\
↪"0x15597db936fc88d8a781ea7da6dce1260a05f10070ab75cd8328659d1343390a\"",
↪"0xDE0B6B3A7640000"]]}]}]}
```


Starting Transactions

```
const io = require('socket.io-client')
const socket = io('http://localhost:5000', {
    path: '/'
})

socket.on('connect', () => {
    console.log("Starting the transactions")
    socket.emit("eth::stop_transactions")//kill any previous transaction logic
    socket.emit("eth::start_transactions","1 0xde0b6b3a7640000")//Start sending the␣
↪transactions
})

socket.open();
```

Note: Any configuration option can be left out, and this entire section can even be null, the example contains all of the defaults.

Ethereum Options

- chainId: The chain id set in the genesis.conf
- networkId: The network id
- difficulty: The initial difficulty set in the genesis.conf file
- initBalance: The initial balance for the accounts
- maxPeers: The maximum number of peers for each node
- gasLimit: The initial gas limit

- homesteadBlock: Set in genesis.conf
- eip155Block: Set in genesis.conf
- eip158Block: Set in genesis.conf

Example (using defaults)

```
{
    "chainId":15468,
    "networkId":15468,
    "difficulty":100000,
    "initBalance":10000000000000000000,
    "maxPeers":1000,
    "gasLimit":4000000,
    "homesteadBlock":0,
    "eip155Block":0,
    "eip158Block":0
}
```

geth

```
whiteblock geth <command> [FLAGS]
```

Geth will allow the user to get infromation and run geth commands.

- **Available SubCommands:**
  - block_listener Get block listener
  - get_accounts Get account information
  - get_balance Get account balance information
  - get_block Get block information
  - get_block_number Get block number
  - get_hash_rate Get hasg rate
  - get_recent_sent_tx Get recently sent transaction
  - get_transaction Get transaction information
  - get_transaction_count Get transaction count
  - get_transaction_receipt Get transaction receipt
  - send_transaction Sends a transaction
  - start_mining Start Mining
  - start_transactions Start transactions
  - stop_mining Stop mining
  - stop_transactions Stop transactions
- **Flags:**
  - -h, –help: help for geth
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

geth block_listener

```
whiteblock geth block_listener [block number] [FLAGS]
```

Get all blocks and continue to subscribe to new blocks

Format: [block number] Params: The block number to start at or None for all blocks Response: Will emit on eth::block_listener for every block after the given block or 0 that exists/has been created

- **Flags:**
  - -h, –help: help for block_listener

geth get_accounts

```
whiteblock geth get_accounts [FLAGS]
```

Get a list of all unlocked accounts

Response: A JSON array of the accounts

- **Flags:**
  - -h, –help: help for get_accounts

geth get_balance

```
whiteblock geth get_balance <address> [FLAGS]
```

Get the current balance of an account

Format: <address> Params: Account address Response: The integer balance of the account in wei

- Flags: - -h, –help: help for get_balance

geth get_block

```
whiteblock geth get_block <block number> [FLAGS]
```

Get the data of a block

Format: <Block Number> Params: Block number

- **Flags:**
  - -h, –help: help for get_block

geth get_block_number

```
whiteblock geth get_block_number [FLAGS]
```

Get the current highest block number of the chain

Response: The block number

- **Flags:**
  - -h, –help: help for get_block_number

geth get_hash_rate

```
whiteblock geth get_hash_rate [FLAGS]
```

Get the current hash rate per node

Response: The hash rate of a single node in the network

- **Flags:**
  - -h, –help: help for get_hash_rate

geth get_recent_sent_tx

```
whiteblock geth get_recent_sent_tx [NUMBER] [FLAGS]
```

Get a number of the most recent transactions sent

Format: [number] Params: The number of transactions to retrieve Response: JSON object of transaction data

- **Flags:**
  - -h, –help: help for get_recent_sent_tx

geth get_transaction

```
whiteblock geth get_transaction <hash> [FLAGS]
```

Get a transaction by its hash

Format: <hash> Params: The transaction hash

Response: JSON representation of the transaction.

- **Flags:**
  - -h, –help: help for get_transaction

geth get_transaction_count

```
whiteblock geth get_transaction_count <address> [BLOCK NUMBER] [FLAGS]
```

Get the transaction count sent from an address, optionally by block

Format: <address> [block number] Params: The sender account, a block number Response: The transaction count

- **Flags:**
  - -h, –help: help for get_transaction_count

geth get_transaction_receipt

```
whiteblock geth get_transaction_receipt <hash> [FLAGS]
```

Get the transaction receipt by the tx hash

Format: <hash> Params: The transaction hash Response: JSON representation of the transaction receipt.

- **Flags:**
  - -h, –help: help for get_transaction_receipt

geth send_transaction

```
whiteblock geth send_transaction <from address> <to address> <gas> <gas price> <value␣
↪to send> [FLAGS]
```

Send a transaction between two accounts

Format: <from> <to> <gas> <gas price> <value> Params: Sending account, receiving account, gas, gas price, amount to send, transaction data, nonce Response: The transaction hash

- Flags: - -h, –help: help for send_transaction

geth start_mining

```
whiteblock geth start_mining [node 1 number] [node 2 number]... [FLAGS]
```

Send the start mining signal to nodes, may take a while to take effect due to DAG generation

Format: [node 1 number] [node 2 number]... Params: A list of the nodes to start mining or None for all nodes Response: The number of nodes which successfully received the signal to start mining

- **Flags:**
  - -h, –help: help for start_mining

geth start_transactions

```
whiteblock geth start_transactions <tx/s> <value> [DESTINATION] [FLAGS]
```

Start sending transactions according to the given parameters, value = -1 means randomize value.

Format: <tx/s> <value> [destination] Params: The amount of transactions to send in a second, the value of each transaction in wei, the destination for the transaction

- **Flags:**
  - -h, –help: help for start_transactions
  - geth stop_mining

geth stop_mining

```
whiteblock geth stop_mining [node 1 number] [node 2 number]... [FLAGS]
```

Send the stop mining signal to nodes

Format: [node 1 number] [node 2 number]... Params: A list of the nodes to stop mining or None for all nodes Response: The number of nodes which successfully received the signal to stop mining

- **Flags:**
  - -h, –help: help for stop_mining

geth stop_transactions

```
whiteblock geth stop_transactions [FLAGS]
```

Stops the sending of transactions if transactions are currently being sent

- **Flags:**
  - -h, –help: help for stop_transactions

Geth (Go-Ethereum)

Note: Any configuration option can be left out, and this entire section can even be null, the example contains all of the defaults

Options

- chainId: The chain id set in the genesis.conf
- networkId: The network id
- difficulty: The initial difficulty set in the genesis.conf file
- initBalance: The initial balance for the accounts
- maxPeers: The maximum number of peers for each node

- gasLimit: The initial gas limit
- homesteadBlock: Set in genesis.conf
- eip155Block: Set in genesis.conf
- eip158Block: Set in genesis.conf

Example (using defaults)

```
{
    "chainId":15468,
    "networkId":15468,
    "difficulty":100000,
    "initBalance":10000000000000000000,
    "maxPeers":1000,
    "gasLimit":4000000,
    "homesteadBlock":0,
    "eip155Block":0,
    "eip158Block":0
}
```

5.10 Syscoin

Syscoin (RegTest)

Options:

- rpcUser: The username credential
- rpcPass: The password credential
- masterNodeConns: The number of connections to set up for the master nodes
- nodeConns: The number of connections to set up for the normal nodes
- percentMasternodes: The percentage of the network consisting of master nodes
- options: Options to set enabled for all nodes
- senderOptions: Options to set enabled for senders
- receiverOptions: Options to set enabled for receivers
- mnOptions: Options to set enabled for master nodes
- extras: Extra options to add to the config file for all nodes
- senderExtras: Extra options to add to the config file for senders
- receiverExtras: Extra options to add to the config file for receivers
- mnExtras: Extra options to add to the config file for master nodes
- **sys::start_test**
  - Description: Start the propogation/tps test for syscoin
  - Params: The max number of test results to retrieve
  - Format: {“waitTime”:<seconds to wait>,”minCompletePercent”:<percentage>,”numberOfTransactions”:<number of tx>}
  - Example:

```
{
    "waitTime":11,
    "minCompletePercent":97.7,
    "numberOfTransactions":500
}
```

- **sys::get_recent_test_results**
  - Description: Get recent test results
  - Params: The max number of test results to retrieve
  - Format: [number]
  - Example: 5
  - Response: Data on the last x test results

```
whiteblock sys <command> [FLAGS]
```

Alias: SYS, syscoin

Sys will allow the user to get infromation and run SYS commands.

- **Available Commands:**
  - test SYS test commands.
- **Flags:**
  - -h, –help : help for sys

sys test

```
whiteblock sys test <command> [FLAGS]
```

Available Commands: results Get results from a previous test. start Starts propagation test.

- **Flags:**
  - -h, –help : help for test

sys test start

```
whiteblock sys test start <wait time> <min complete percent> <number of tx> [FLAGS]
```

Sys test start will start the propagation test. It will wait for the signal start time, have nodes send messages at the same time, and require to wait a minimum amount of time then check receivers with a completion rate of minimum completion percentage.

Format: <wait time> <min complete percent> <number of tx> Params: Time in seconds, percentage, number of transactions

- **Flags:**
  - -h, –help : help for start
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

sys test results

```
whiteblock sys test results <test number> [FLAGS]
```

Sys test results pulls data from a previous test or tests and outputs as csv.

Format: <test number> Params: Test number

- **Flags:**
  - -h, –help : help for results
  - -a, –server-addr string: server address with port 5000 (default “localhost:5000”)

Example (using defaults)

```
{
    "rpcUser":"username",
    "rpcPass":"password",
    "masterNodeConns":25,
    "nodeConns":8,
    "percentMasternodes":90,
    "options":[
        "server",
        "regtest",
        "listen",
        "rest"
    ],
    "senderOptions":[
        "tpstest",
        "addressindex"
    ],
    "mnOptions":[],
    "receiverOptions":[
       "tpstest"
    ],
    "extras":[],
    "senderExtras":[],
    "receiverExtras":[],
    "mnExtras":[]
}
```