Tolar gRPC client API documentation

TOLAR GRPC CLIENT API DOCUMENTATION

Document change log

Document version	Change author(s)	Date	Document changes
1.0	Iva Brajer	17/05/2019	• Initial document
1.1	Iva Brajer	30/05/2019	• Chapter 1.3. List addresses: fixed typo and changed string to bytes for addresses • Added chapter 1.8. List balance per address • Added chapter 1.9. Send raw transaction
1.2	Iva Brajer	30/05/2019	• Chapters 1.4. Verify Address, 1.5. Create new address, 1.6. Export key file: fixed typo and changed string to bytes
1.3	Iva Brajer	04/06/2019	• Added response code descriptions for every API call • Added chapter 4.5. GetTransactionList • Added confirmation_timestamp to chapters 4.2. GetBlock, 4.3. GetTransaction, 4.5. GetTransactionList
1.4	Iva Brajer	06/06/2019	• Added chapters 1.10. ChangePassword and 1.11. ChangeAddressPassword
1.5	Dario Pažin	31/07/2019	• Chapter 1.9. SendRawTransaction: change in response parameter. API call now returns [transaction_hash: bytes] instead of [result: boolean]
1.6	Iva Brajer	28/08/2019	• Chapter 4.3. GetTransaction: updated response type • Chapter 4.5. GetTransactionList: updated TransactionDetails custom message type
1.7	Iva Brajer	06/09/2019	• Chapter 1.8. ListBalancePerAddress: balance type change • Chapter 1.9. SendRawTransaction: amount type change • Chapter 2.3. GetBalance: balance type change
1.8	Iva Brajer	25/09/2019	• Chapter 1.9. SendRawTransaction: added missing additional fields with extended example • Chapter 2.1. SendTransaction: updated request custom messages • Chapter 2.2. ListUnspentOutputs removed • Chapter 2.2. GetNonce added instead
1.9.	Dario Pažin	14/10/2019	• Added chapters: • 1.12. Send fund transfer transaction added • 1.13. Send deploy contract transaction • 1.14. Send execute function transaction • 2.4. Try call transaction • Chapter 4.3. Get transaction updated response parameters • Chapter 4.5. Get transaction list updated response parameters
1.10.	Igor Jerkovi	08/11/2019	• Chapter 4.2. GetBlock is now renamed to GetBlockByHash • Added chapters: • 4.3. GetBlockByIndex
1.11.	Dario Pažin	22/11/2019	• Added chapters: • 5.1. GetBlockInfoByHash • 5.2. GetBlockInfoByIndex • 5.3. GetLatestBlocks • 5.4. GetTransaction • 5.5. GetAddress

1.12.	Dario Pažin	28/11/2019	• Chapter 2.1. SendSignedTransaction: change in response parameter. API call now returns [transaction_hash: bytes] instead of [result: boolean]
1.13.	Igor Jerkovi	02/12/2019	• Removed chapters: • 2.2. Get Nonce • 2.3. Get Balance • 2.4. Try call transaction • Added chapters • 4.7. Get Nonce • 4.8. Get Balance • 4.9. Get Latest Balance • 4.10. Try call transaction

Introduction

This document guides through gRPC client APIs available by Tolar HashNet master or light node. APIs can roughly be separated into several categories:

- account management (e.g. create new address, list existing addresses)
- transactions (e.g. send new transaction)
- network information (e.g. get peer count, check if master node)
- block explorer (e.g. get confirmed transaction, get confirmed block, get balance for address)

APIs are available on two separate endpoints (different local ports):

1. Client endpoint: transactions + block explorer + network information
2. Account endpoint: account management

Endpoints are defined in master or light node configuration files.

gRPC protocol buffer schemas

In order to be able to use gRPC client APIs, as a first step protobuf schemas should be provided by Tolar HashNet team. With provided schemas, it’s possible to run protocol buffer compiler (protoc tool) for desired language. Generated classes could be used to send request messages and processes response messages from Tolar HashNet APIs.

Tolar address format

Tolar address is formatted in the following way:

Capital letter T (1 byte) – unique address space (20 bytes) – checksum part (4 bytes)

Description of possible API responses

Code	Number	Description	Closest HTTP Mapping
OK	0	Not an error; returned on success.	200 OK
CANCELLED	1	The operation was cancelled, typically by the caller.	499 Client Closed Request
UNKNOWN	2	Unknown error. For example, this error may be returned when a `Status` value received from another address space belongs to an error space that is not known in this address space. Also errors raised by APIs that do not return enough error information may be converted to this error.	500 Internal Server Error
INVALID_ARGUMENT	3	The client specified an invalid argument. Note that this differs from `FAILED_PRECONDITION`. `INVALID_ARGUMENT` indicates arguments that are problematic regardless of the state of the system (e.g., a malformed file name).	400 Bad Request
DEADLINE_EXCEEDED	4	The deadline expired before the operation could complete. For operations that change the state of the system, this error may be returned even if the operation has completed successfully. For example, a successful response from a server could have been delayed long	504 Gateway Timeout

NOT_FOUND	5	Some requested entity (e.g., file or directory) was not found. Note to server developers: if a request is denied for an entire class of users, such as gradual feature rollout or undocumented whitelist, `NOT_FOUND` may be used. If a request is denied for some users within a class of users, such as user-based access control, `PERMISSION_DENIED` must be used.	404 Not Found
ALREADY_EXISTS	6	The entity that a client attempted to create (e.g., file or directory) already exists.	409 Conflict
PERMISSION_DENIED	7	The caller does not have permission to execute the specified operation. `PERMISSION_DENIED` must not be used for rejections caused by exhausting some resource (use `RESOURCE_EXHAUSTED` instead for those errors). `PERMISSION_DENIED` must not be used if the caller can not be identified (use `UNAUTHENTICATED` instead for those errors). This error code does not imply the request is valid or the requested entity exists or satisfies other pre-conditions.	403 Forbidden
UNAUTHENTICATED	16	The request does not have valid authentication credentials for the operation.	401 Unauthorized
RESOURCE_EXHAUSTED	8	Some resource has been exhausted, perhaps a per-user quota, or perhaps the entire file system is out of space.	429 Too Many Requests
FAILED_PRECONDITION	9	The operation was rejected because the system is not in a state required for the operation's execution. For example, the directory to be deleted is non-empty, an rmdir operation is applied to a non-directory, etc. Service implementors can use the following guidelines to decide between `FAILED_PRECONDITION`, `ABORTED`, and `UNAVAILABLE`: (a) Use `UNAVAILABLE` if the client can retry just the failing call. (b) Use `ABORTED` if the client should retry at a higher level (e.g., when a client-specified test-and-set fails, indicating the client should restart a read-modify-write sequence). (c) Use `FAILED_PRECONDITION` if the client should not retry until the system state has been explicitly fixed. E.g., if an "rmdir" fails because the directory is non-empty, `FAILED_PRECONDITION` should be returned since the client should not retry unless the files are deleted from the directory.	400 Bad Request
ABORTED	10	The operation was aborted, typically due to a concurrency issue such as a sequencer check failure or transaction abort. See the guidelines above for deciding between `FAILED_PRECONDITION`, `ABORTED`, and `UNAVAILABLE`.	409 Conflict
OUT_OF_RANGE	11	The operation was attempted past the valid range. E.g., seeking or reading past end-of-file. Unlike `INVALID_ARGUMENT`, this error indicates a problem that may be fixed if the system state changes. For example, a 32-bit file system will generate `INVALID_ARGUMENT` if asked to read at an offset that is not in the range [0,2^32-1], but it will generate `OUT_OF_RANGE` if asked to read from an offset past the current file size. There is a fair bit of overlap between `FAILED_PRECONDITION` and `OUT_OF_RANGE`. We recommend using `OUT_OF_RANGE` (the more specific error) when it applies so that callers who are iterating through a space can easily look for an `OUT_OF_RANGE` error to detect when they are done.	400 Bad Request
UNIMPLEMENTED	12	The operation is not implemented or is not supported/enabled in this service.	501 Not Implemented
INTERNAL	13	Internal errors. This means that some invariants expected by the underlying system have been broken. This error code is reserved for serious errors.	500 Internal Server Error
UNAVAILABLE	14	The service is currently unavailable. This is most likely a transient condition, which can be corrected by retrying with a backoff. Note that it is not always safe to retry non-idempotent operations.	503 Service Unavailable
DATA_LOSS	15	Unrecoverable data loss or corruption.	500 Internal Server Error

1. Account management APIs

1.1. Create

Endpoint	Account
Method name	Create
Method description	Creates new keystore attached to master or light node
Request parameters	[master_password: string] Locks entire keystore with this password if provided, if empty keystore will not be locked
Response parameters	[result: boolean] True if keystore was successfully created, false otherwise
Response code descriptions	• OK – request successful • ALREADY_EXISTS – keystore already created or open • NOT_FOUND – keystore does not exist
Faulty scenarios	Not able to create another keystore if there is already existing one

Example	Request protobuf message:
	
	{
	“master_password”: “VerySafePassword”
	}
	
	Response protobuf message:
	
	{
	“result”: true
	}

1.2. Open

Endpoint	Account
Method name	Open
Method description	Opens existing keystore attached to master or light node
Request parameters	[master_password: string] Unlocks keystore with this password if keystore was originally locked with provided password
Response code descriptions	• OK – request successful • ALREADY_EXISTS – keystore already open • NOT_FOUND – keystore does not exist • PERMISSION_DENIED – invalid password to unlock keystore
Response parameters	[result: boolean] True if keystore was successfully opened, false otherwise
Faulty scenarios	Not able to open keystore if not previously created or if wrong master password was provided

Example	Request protobuf message:
	
	{
	“master_password”: “VerySafePassword”
	}
	
	Response protobuf message:
	
	{
	“result”: true
	}

1.3. List addresses

Endpoint	Account
Method name	ListAddresses
Method description	List all addresses in keystore attached to master or light node
Request parameters	None
Response parameters	[addresses: bytes array] Array of addresses in Tolar address format
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore
Faulty scenarios	Keystore not found or not previously opened/unlocked

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“addresses”: [“54948c78114bc39675157e097830ae63c0da7857a19c13aec7”, “54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”]
	}

1.4. Verify address

Endpoint	Account
Method name	VerifyAddress
Method description	Verifies if provided address string is in valid Tolar address format

Request parameters	[address: bytes] Address in hex string format
Response parameters	[result: boolean] True if provided address is in valid Tolar address format, false otherwise
Response code descriptions	• OK – request successful
Faulty scenarios	Keystore not found or not previously opened/unlocked

Example	Request protobuf message:
	
	{
	“address”: “abcdef123456”
	}
	
	Response protobuf message:
	
	{
	“result”: “false”
	}

1.5. Create new address

Endpoint	Account
Method name	CreateNewAddress
Method description	Creates new address in keystore attached to master or light node
Request parameters	[name: string] Optional address description name [lock_password: string] Optional password to protect generate keypair for newly created address [lock_hint: string] Optional password hint for selected password
Response parameters	[address: bytes] If successfully created, return newly created address in Tolar address format
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore
Faulty scenarios	Keystore not found or not previously opened/unlocked

Example	Request protobuf message:
	
	{
	“name”: “NewAddress”,
	“lock_password”: “pass123”
	}
	
	Response protobuf message:
	
	{
	“address”: “ 54948c78114bc39675157e097830ae63c0da7857a19c13aec7”
	}

1.6. Export key file

Endpoint	Account
Method name	ExportKeyFile
Method description	Exports key file for selected address from keystore attached to master or light node
Request parameters	[address: bytes] Selected address for which export keypair information is required
Response parameters	[json_key_file: string] If successful, return key file in encrypted JSON format
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore
Faulty scenarios	Keystore not found or not previously opened/unlocked Address does not exist in keystore

Example	Request protobuf message:
	{
	“address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”
	}
	Response protobuf message:
	{
	“json_key_file”: “{
	"address" : "f9f02416d894487e7bbd9d74065f7996cbdbf52b",
	"crypto" : {
	"cipher" : "aes-128-ctr",
	"cipherparams" : {"iv" : "28fe2f484412dcdc1e2c56544e511d1c"},
	"ciphertext" :
	"db10f6e015eb7d744a8de7a2ab2a97f4542c60cb48b846d441ae4add00b8a469",
	"kdf" : "scrypt",
	"kdfparams" : {
	"dklen" : 32,
	"n" : 262144,
	"p" : 1,
	"r" : 8,
	"salt" :
	"68caf683e20ae150d7f2150c25426caf178c2f2ee9082cfa784239838ae64b68"
	},
	"mac" : "86006944babe7d7d80c08c29cd3defc7aebe1fd9bdc9d3aee2cb8f6382982d6e"
	},
	"id" : "32addc9f-8942-93e9-f109-f6fa8776fdf1",
	"version" : 3
	}”
	}

1.7. Import key file

Endpoint	Account
Method name	ImportKeyFile
Method description	Imports key file to keystore attached to master or light node

Request parameters	[json_key_file: string] Key file in encrypted JSON format [name: string] Optional name for imported address [lock_password: string] Provide lock password if original key file was password protected [lock_hint: string] Optional lock hint for lock password
Response parameters	[result: boolean] Returns true if import successful, false otherwise
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore
Faulty scenarios	Keystore not found or not previously opened/unlocked Key file can’t be unlocked with provided password

Example	Request protobuf message:
	
	{
	“json_key_file”: “{
	"address" : "f9f024
	Balance on sender address is not enough to send requested amount of tolars
	
	16d894487e7bbd9d74065f7996cbdbf52b",
	"crypto" : {
	"cipher" : "aes-128-ctr",
	"cipherparams" : {"iv" : "28fe2f484412dcdc1e2c56544e511d1c"},
	"ciphertext" :
	"db10f6e015eb7d744a8de7a2ab2a97f4542c60cb48b846d441ae4add00b8a469",
	"kdf" : "scrypt",
	"kdfparams" : {
	"dklen" : 32,
	"n" : 262144,
	"p" : 1,
	"r" : 8,
	"salt" :
	"68caf683e20ae150d7f2150c25426caf178c2f2ee9082cfa784239838ae64b68"
	},
	"mac" : "86006944babe7d7d80c08c29cd3defc7aebe1fd9bdc9d3aee2cb8f6382982d6e"
	},
	"id" : "32addc9f-8942-93e9-f109-f6fa8776fdf1",
	"version" : 3
	}”
	}
	
	Response protobuf message:
	
	{
	“result”: “true”
	}

1.8. List balance per address

Endpoint	Account
Method name	ListBalancePerAddress

Method description	List all addresses (stored in keystore attached to master or light node) with their associated name and current balance status
Request parameters	None
Response parameters	[addresses: AddressBalance array] Array of addresses paired with their name and balance
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore
Custom messages	[AddressBalance] - [addres: bytes] Tolar address - [balance: bytes] Current balance for that address (in tolars) - [address_name: string] Associated address name (if exists)
Faulty scenarios	Keystore not found or not previously opened/unlocked

Example	Request protobuf message:
	
	{
	
	}
	
	Response protobuf message:
	
	{
	
	“addresses”: [{
	
	“address“: “54948c78114bc39675157e097830ae63c0da7857a19c13aec7”,
	
	“name”: “”,
	
	“balance”: 1570},
	
	{“address”: “54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”,
	
	“name”: “CustomAddress”,
	
	“balance”: 124]}
	
	]

1.9. Send raw transaction

Endpoint	Account
Method name	SendRawTransaction
Method description	Sends data for creating transaction on light node only if sender address private key is stored in node keystore, transaction signing is left for node to handle

Request parameters	[sender_address: bytes] Address in Tolar format from which transaction will be send [receiver_address: bytes] Address in Tolar format to which transaction will be send [amount: bytes] Amount of tolars to send [sender_address_password: string] Password to unlock private key for sender address on node keystore (leave empty for no password) [gas: bytes] Maximum gas (gas limit) that will be spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) [data: string] Smart contract bytecode in hex format [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce)
Response parameters	[transaction_hash: bytes] Transaction hash
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore • INVALID_ARGUMENT – something wrong with request parameter(s) • ABORTED – keystore not able to sign transaction
Faulty scenarios	Keystore not found or not previously opened/unlocked Sender address is not found in node keystore Password for sender address is not correct and private key can’t be accessed Sender address balance is not enough to send requested amount of tolars

Example	Request protobuf message:
	
	{
	“sender_address”: “ 54948c78114bc39675157e097830ae63c0da7857a19c13aec7”,
	“receiver_address”:” 54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”,
	“amount”: 100,
	“sender_address_password”: “pass123”,
	"gas": 21000,
	"gas_price": 1,
	"data": "",
	"nonce": 0
	}
	
	Response protobuf message:
	
	{
	“transaction_hash”: “c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	}

1.10. Change password

Endpoint	Account
Method name	ChangePassword
Method description	Changes master password used to lock entire keystore
Request parameters	[old_master_password: string] Current master password used to lock keystore [new_master_password: string] New master password that will replace current one
Response parameters	[result: bool] True if password change was successful, false otherwise
Response code descriptions	• OK – request successful • PERMISSION_DENIED – provided old master password was not able to unlock keystore • NOT_FOUND – keystore does not exist
Faulty scenarios	Keystore not found or not previously opened/unlocked Provided old master password is invalid

Example	Request protobuf message:
	
	{
	“old_master_password”: “ old”,
	“new_master_password”:” new”
	}
	
	Response protobuf message:
	
	{
	“result”: “true”
	}

1.11. Change address password

Endpoint	Account
Method name	ChangeAddressPassword
Method description	Changes lock password for single address used to lock its private key in keystore
Request parameters	[address: bytes] Address for which password changing is required [old_password: string] Current address password [new_password: string] New address password that will replace current one
Response parameters	[result: bool] True if password change was successful, false otherwise
Response code descriptions	• OK – request successful • PERMISSION_DENIED – provided old password was not able to unlock address private key in keystore • NOT_FOUND – keystore does not exist
Faulty scenarios	Keystore not found or not previously opened/unlocked Provided old password is invalid

Example	Request protobuf message:
	
	{
	“address”: “54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”,
	“old_master_password”: “ old”,
	“new_master_password”:” new”
	}
	
	Response protobuf message:
	
	{
	“result”: “true”
	}

1.12. Send fund transfer transaction

Endpoint	Account
Method name	SendFundTransferTransaction
Method description	Sends data for creating transaction on light node only if sender address private key is stored in node keystore, transaction signing is left for node to handle. Transaction used for transferring funds from sender to receiver address.
	[sender_address: bytes] Address in Tolar format from which transaction will be send [receiver_address: bytes] Address in Tolar format to which transaction will be send [amount: bytes] Amount of tolars to send [sender_address_password: string] Password to unlock private key for sender address on node keystore (leave empty for no password) [gas: bytes] Maximum gas (gas limit) that will be spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce)
Response parameters	[transaction_hash: bytes] Transaction hash

Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore • INVALID_ARGUMENT – something wrong with request parameter(s) • ABORTED – keystore not able to sign transaction
Faulty scenarios	Keystore not found or not previously opened/unlocked Sender address is not found in node keystore Password for sender address is not correct and private key can’t be accessed Sender address balance is not enough to send requested amount of tolars

Example	Request protobuf message:
	
	{
	“sender_address”: “ 54948c78114bc39675157e097830ae63c0da7857a19c13aec7”,
	“receiver_address”:” 54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”,
	“amount”: 100,
	“sender_address_password”: “pass123”,
	"gas": 21000,
	"gas_price": 1,
	"nonce": 0
	}
	
	Response protobuf message:
	
	{
	“transaction_hash”: “c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	}

1.13. Send deploy contract transaction

Endpoint	Account
Method name	SendDeployContractTransaction
Method description	Sends data for creating transaction on light node only if sender address private key is stored in node keystore, transaction signing is left for node to handle. Transaction used for deploying the contract.

Request parameters	[sender_address: bytes] Address in Tolar format from which transaction will be send [amount: bytes] Amount of tolars (can be required by contract constructor) [sender_address_password: string] Password to unlock private key for sender address on node keystore (leave empty for no password) [gas: bytes] Maximum gas (gas limit) that will be spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) [data: string] Smart contract bytecode in hex format [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce)
Response parameters	[transaction_hash: bytes] Transaction hash
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore • INVALID_ARGUMENT – something wrong with request parameter(s) • ABORTED – keystore not able to sign transaction
Faulty scenarios	Keystore not found or not previously opened/unlocked Sender address is not found in node keystore Not enough gas to deploy contract Sender address balance is not enough to send requested amount of tolars

Example	Request protobuf message:
	
	{
	“sender_address”: “ 54948c78114bc39675157e097830ae63c0da7857a19c13aec7”,
	“amount”: 0,
	“sender_address_password”: “pass123”,
	"gas": 200000,
	"gas_price": 1,
	"data": "6080604052341561000f57600080fd5b60b98061001d6000396000f300"
	"608060405260043610603f576000357c0100000000000000000000000000"
	"00000000000000000000000000000000900463ffffffff168063b3de64"
	"8b146044575b600080fd5b3415604e57600080fd5b606a600480360381"
	"0190808035906020019092919050505060805650b604051808281526020"
	"0191505060405180910390f35b6000600782029050919050505600a16562"
	"7a7a72305820f294e834212334e2978c6dd090355312a3f0f9476b8eb9"
	"8fb480406fc2728a960029",
	"nonce": 0
	}
	
	Response protobuf message:
	
	{
	“transaction_hash”: “c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	}

1.14. Send execute function transaction

Endpoint	Account
Method name	SendExecuteFunctionTransaction
Method description	Sends data for creating transaction on light node only if sender address private key is stored in node keystore, transaction signing is left for node to handle. Transaction used for executing contract functions

Request parameters	[sender_address: bytes] Address in Tolar format from which transaction will be send [receiver_address: bytes] Contract address in Tolar format [amount: bytes] Amount of tolars if needed in contract function [sender_address_password: string] Password to unlock private key for sender address on node keystore (leave empty for no password) [gas: bytes] Maximum gas (gas limit) that will be spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) [data: string] Function bytecode in hex format [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce)
Response parameters	[transaction_hash: bytes] Transaction hash
Response code descriptions	• OK – request successful • PERMISSION_DENIED – not able to access keystore • INVALID_ARGUMENT – something wrong with request parameter(s) • ABORTED – keystore not able to sign transaction
Faulty scenarios	Keystore not found or not previously opened/unlocked Sender address is not found in node keystore Not enough gas to execute contract function Sender address balance is not enough to send requested amount of tolars

Example	Request protobuf message:
	
	{
	“sender_address”: “ 54948c78114bc39675157e097830ae63c0da7857a19c13aec7”,
	“receiver_address”:” 54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”,
	“amount”: 100,
	“sender_address_password”: “pass123”,
	"gas": 210000,
	"gas_price": 1,
	"data": "0000000000000000000000000000000000000000000000000000000000000001",
	"nonce": 0
	}
	
	Response protobuf message:
	
	{
	“transaction_hash”: “c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	}

2. Transaction APIs

2.1. Send signed transaction

Endpoint	Client
Method name	SendSignedTransaction
Method description	Send signed transaction with prepared transaction inputs and outputs
Request parameters	[transaction: SignedTransaction] Signed Transaction message with signed Input messages and raw Output messages, client keypair should be used for signing
Response code descriptions	• OK – request successful • INTERNAL – failed to process transaction

Custom messages	[SignedTransaction] - [Transaction] ▪ [sender_address: bytes] Address in Tolar format from which transaction will be send ▪ [receiver_address: bytes] Address in Tolar format to which transaction will be send ▪ [amount: bytes] Amount of tolars to send ▪ [sender_address_password: string] Password to unlock private key for sender address on node keystore (leave empty for no password) ▪ [gas: bytes] Maximum gas (gas limit) that will be spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) ▪ [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) ▪ [data: string] Smart contract bytecode in hex format ▪ [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) - [SignatureData] ▪ [hash: bytes] Hash for SignedTransaction ▪ [signature: bytes] Signature for SignedTransaction ▪ [signer_id: bytes] Signer identification for SignedTransaction
Response parameters	[transaction_hash: bytes] Transaction hash
Faulty scenarios	Not able to verify signature of transaction Invalid transaction nonce

Example	Request protobuf message (see chapter 1.9. SendRawTransaction for specific fields examples):
	
	{
	“transaction”: <signed_bytes>
	}
	
	Response protobuf message:
	
	{
	“result”: “true”
	}

3. Network information APIs

3.1. Peer count

Endpoint	Client
Method name	PeerCount
Method description	Get current peer count in running HashNet network
Request parameters	None
Response parameters	[count: uint64] Number of discovered peers in HashNet network
Response code descriptions	• OK – request successful
Faulty scenarios	Peer to peer discovery failure

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“count”: 100
	}

3.2. Master node count

Endpoint	Client
Method name	MasterNodeCount
Method description	Get current master nodes count in running HashNet network

Request parameters	None
Response parameters	[count: uint64] Number of master nodes in HashNet network
Response code descriptions	• OK – request successful
Faulty scenarios	Peer to peer discovery failure

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“count”: 10
	}

3.3. Is master node

Endpoint	Client
Method name	IsMasterNode
Method description	Check if currently pinging master node endpoint in running HashNet network
Request parameters	None
Response parameters	[is_master: bool] Returning true confirms that this is master node
Response code descriptions	• OK – request successful
Faulty scenarios	Peer to peer discovery failure

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“is_master”: true
	}

3.4. Maximum peer count

Endpoint	Client
Method name	MaxPeerCount
Method description	Gets maximum allowed peer count in running HashNet network
Request parameters	None
Response parameters	[count: uint64] Returns maximum allowed peer count
Response code descriptions	• OK – request successful
Faulty scenarios	Peer to peer discovery failure

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“count”: 1000
	}

4. Block explorer APIs

4.1. Get block count

Endpoint	Client
Method name	GetBlockCount
Method description	Gets number of confirmed blocks in current node block chain
Request parameters	None
Response parameters	[block_count: uint64] Number of available confirmed blocks in block chain
Response code descriptions	• OK – request successful
Faulty scenarios	No confirmed blocks due to node malfunction or gossip protocol failure

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“block_count”: 148
	}

4.2. Get block by hash

Endpoint	Client
Method name	GetBlockByHash
Method description	Retrieves confirmed block information from current node block chain
Request parameters	[block_hash: bytes] Hash for requested block
Response parameters	[block_index: uint64] Block index in current block chain [previous_block_hash: bytes] Block hash for previous block in block chain attached to this block [transaction_hashes: bytes array] Array of transaction hashes contained in this block [confirmation_timestamp: uint64] Time when block was confirmed in UNIX timestamp format
Response code descriptions	• OK – request successful • NOT_FOUND – block is not found
Faulty scenarios	Block hash doesn’t exist in block chain Block is not yet confirmed in block chain

Example	Request protobuf message:
	
	{
	“block_hash”: “ c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	}
	
	Response protobuf message:
	
	{
	“block_index”: 9647,
	“previous_block_hash”: “ ae78000220d4a1a6d2b3ca9b14174505d9e4b081e06e1a5f2e79052a2f6e26b8”,
	“next_block_hash”: “392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d”,
	“transaction_hashes”: [“f58ffec7eb33908a32aa4c0c1ec4b30abc2dd9f0dc4da390f46f6b56762fdf24”,
	“0e5669f90fdf46baef98b629efc1e7b461b4f092600a07a5449b963a3865483e”,
	“23795ebb10fc32524e2280734087fe99fe8d5f28db360bbf635a6abe44c872da”],
	“confirmation_timestamp”: 1559653728
	}

4.3. Get block by index

Endpoint	Client
Method name	GetBlockByIndex
Method description	Retrieves confirmed block information from current node block chain
Request parameters	[block_index: uint64] Block index for requested block
Response parameters	[block_index: uint64] Block index in current block chain [previous_block_hash: bytes] Block hash for previous block in block chain attached to this block [transaction_hashes: bytes array] Array of transaction hashes contained in this block [confirmation_timestamp: uint64] Time when block was confirmed in UNIX timestamp format
Response code descriptions	• OK – request successful • NOT_FOUND – block is not found
Faulty scenarios	Block index larger then last confirmed block index exist in block chain

Example	Request protobuf message:
	
	{
	“block_index”: 9467
	}
	
	Response protobuf message:
	
	{
	“block_index”: 9647,
	“previous_block_hash”: “ ae78000220d4a1a6d2b3ca9b14174505d9e4b081e06e1a5f2e79052a2f6e26b8”,
	“next_block_hash”: “392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d”,
	“transaction_hashes”: [“f58ffec7eb33908a32aa4c0c1ec4b30abc2dd9f0dc4da390f46f6b56762fdf24”,
	“0e5669f90fdf46baef98b629efc1e7b461b4f092600a07a5449b963a3865483e”,
	“23795ebb10fc32524e2280734087fe99fe8d5f28db360bbf635a6abe44c872da”],
	“confirmation_timestamp”: 1559653728
	}

4.4. Get transaction

Endpoint	Client
Method name	GetTransaction
Method description	Retrieves confirmed transaction information from current node block chain
Request parameters	[transaction_hash: bytes] Hash for requested transaction

Response parameters	[block_hash: bytes] Block hash of confirmed block where this transaction is found inside [transaction_index: uint64] Index of transaction inside block [sender_address: bytes] Address that initiated this transaction [receiver_address: bytes] Address that received this transaction [value: bytes] Amount send in transaction (for transfer fund) [gas: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) [data: string] Smart contract bytecode in hex format [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) [gas_used: bytes] Gas amount used executing transaction [gas_refunded: bytes] Gas amount that is refunded to sender address after executing transaction [new_address: bytes] New address that is created after executing transaction (deployed contract address) [output: string] The returned data of the call, e.g. a smart contract functions return value [excepted: bool] true if exception happened, false if transaction execution was successful [confirmation_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format
Response code descriptions	• OK – request successful • NOT_FOUND – transaction is not found
Faulty scenarios	Transaction doesn’t exist in any confirmed block

Example	Request protobuf message:
	
	{
	
	“transaction_hash”: “ c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	
	}
	
	Response protobuf message:
	
	{
	
	“block_hash”: “ ae78000220d4a1a6d2b3ca9b14174505d9e4b081e06e1a5f2e79052a2f6e26b8”,
	
	“transaction_index”: 9568,
	
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	
	“receiver_address”: “”,
	
	"value": 0,
	
	“gas": 105000,
	
	"gas_price": 1,
	
	"data": ""b3de648b00000000000000000000000000000000000000000000000000000000000000001"",
	
	"gas_used": 21730
	
	"gas_refunded": 0
	
	"new_address": "5456a09d5c06e23ec6a71a7db606549ec4bde1788c71a9552b"
	
	"output": "0000000000000000000000000000000000000000000000000000000000000007"
	
	"excepted": false
	
	“confirmation_timestamp”: 1559653728
	
	}

4.5. Get blockchain information

Endpoint	Client
Method name	GetBlockchainInfo
Method description	Retrieves block chain statistics information for current node block chain
Request parameters	None
Response parameters	[confirmed_blocks_count: uint64] Total number of confirmed blocks in current node block chain [total_block_count: uint64] Total number of blocks in current block chain (confirmed + pending) [last_confimed_block_hash: bytes] Hash of latest confirmed block in current block chain
Response code descriptions	• OK – request successful
Faulty scenarios	Block chain is empty due to node malfunction or gossip protocol failure

Example	Request protobuf message:
	
	{
	}
	
	Response protobuf message:
	
	{
	“confirmed_blocks_count”: 560,
	“total_blocks_count”: 671,
	“last_confirmed_block_hash”: “1da0d7aeff8773579899fa48ab8bc6a72503240f684375c6123c197b2cb863ea”
	}

4.6. Get transaction list

Endpoint	Client
Method name	GetTransactionList
Method description	Retrieves most recent transaction list based on transaction limit and how many transactions to skip (provides ability to get transactions in batches)
Request parameters	[addresses: bytes array] List of all addresses by which transaction should be filtered (leave empty to apply no filter and return all transactions) [limit: uint64] Maximum number of transactions to return in one batch (no more than 1000) [skip: uint64] Number of most recent transactions to skip starting from blockchain’s last confirmed block
Response parameters	[transactions: TransactionDetails array] List of all recent transactions filtered by addresses
Response code descriptions	• OK – request successful • INVALID_ARGUMENT – request parameter(s) invalid

Custom messages	[TransactionDetails] - [block_hash: bytes] Block hash of confirmed block where this transaction is found inside - [transaction_index: uint64] Index of transaction inside block - [sender_address: bytes] Address that initiated this transaction - [receiver_address: bytes] Address that received this transaction - [value: bytes] Amount send in transaction (for transfer fund) - [gas: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) - [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) - [data: string] Smart contract bytecode in hex format - [gas_used: bytes] Gas amount used executing transaction - [gas_refunded: bytes] Gas amount that is refunded to sender address after executing transaction - [new_address: bytes] New address that is created after executing transaction (deployed contract address) - [output: string] The returned data of the call, e.g. a smart contract functions return value - [excepted: bool] true if exception happened, false if transaction execution was successful - [confirmation_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format
Faulty scenarios	Block chain is empty due to node malfunction or gossip protocol failure Expected limit is more than 1000 transactions Address to filter by is not in Tolar address format

Example

Request protobuf message:

```
{
“addresses”: [],
“limit”: 2,
“skip”: 0
}
```

Response protobuf message:

```
{
“transactions”: [{
“block_hash”: “ ae78000220d4a1a6d2b3ca9b14174505d9e4b081e06e1a5f2e79052a2f6e26b8”,
“transaction_index”: 9568
“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
"value": 156,
"gas": 21000,
"gas_price": 1,
"gas_used": 21730
"gas_refunded": 0
"new_address": "5456a09d5c06e23ec6a71a7db606549ec4bde1788c71a9552b"
"output": "0000000000000000000000000000000000000000000000000000000000000009"
"excepted": false
“confirmation_timestamp”: 1559653728},
{“block_hash”: “ ae78000220d4a1a6d2b3ca9b14174505d9e4b081e06e1a5f2e79052a2f6e26b8”,
“transaction_index”: 9567
“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
"value": 12360,
"gas": 21000,
"gas_price": 1,
"gas_used": 21730
"gas_refunded": 0
"new_address": "5456a09d5c06e23ec6a71a7db606549ec4bde1788c71a9552b"
"output": "0000000000000000000000000000000000000000000000000000000000000007"
"excepted": false
“confirmation_timestamp”: 1559653739}
]}
```

4.7. Get nonce

Endpoint	Client
Method name	GetNonce
Method description	Get next available nonce value for specific address
Request parameters	[address: bytes] Selected address in Tolar address format
Response parameters	[nonce:bytes] Next available nonce value
Response code descriptions	• OK – request successful • INVALID_ARGUMENT – request parameter invalid
Faulty scenarios	Requesting nonce for invalid Tolar address

Example	Request protobuf message:
	
	{
	“address”: “ 54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”
	}
	
	Response protobuf message:
	
	{
	“nonce": 12
	}

4.8. Get balance

Endpoint	Client
Method name	GetBalance
Method description	Get current balance (of tolars) for selected address
Request parameters	[address: bytes] Selected address in Tolar address format
Response parameters	[balance: bytes] Balance expressed in tolars count
Response code descriptions	• OK – request successful • INVALID_ARGUMENT – request parameter invalid
Faulty scenarios	Requesting balance for non-existing address

Example	Request protobuf message:
	
	{
	“address”: “ 54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”
	}
	
	Response protobuf message:
	
	{
	“balance”: 45
	}

4.9. Get latest balance

Endpoint	Client
Method name	GetLatestBalance
Method description	Get current balance (of tolars) for selected address
Request parameters	[address: bytes] Selected address in Tolar address format
Response parameters	[balance: bytes] Balance expressed in tolars count
Response code descriptions	• OK – request successful • INVALID_ARGUMENT – request parameter invalid
Faulty scenarios	Requesting balance for non-existing address

Example	Request protobuf message:
	
	{
	“address”: “ 54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”
	}
	
	Response protobuf message:
	
	{
	“balance”: 45
	}

4.10. Try call transaction

Endpoint	Client

Method name	TryCallTransaction
Method description	Executes read only contract functions on evm without spending gas or having any effect to address balances.
Request parameters	[sender_address: bytes] Address in Tolar format from which transaction will be send [receiver_address: bytes] Contract address in Tolar format [amount: bytes] Amount of tolars if needed in contract function [gas: bytes] Maximum gas (gas limit) that is available for call function [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) [data: string] Contract function bytecode in hex format [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce)
Response parameters	[output: string] The returned data of the call, e.g. a smart contract functions return value
Response code descriptions	• OK – request successful • INVALID_ARGUMENT – something wrong with request parameter(s)
Faulty scenarios	Not enough gas to execute contract function

Example	Request protobuf message:
	
	{
	
	“sender_address”: “ 54948c78114bc39675157e097830ae63c0da7857a19c13aec7”,
	
	“receiver_address”:” 54949f54114bc39675157e123830ae7a70da7adfa19c24c8db”,
	
	“amount”: 0,
	
	"gas": 210000,
	
	"gas_price": 1,
	
	"data": "0000000000000000000000000000000000000000000000000000000000000001",
	
	"nonce": 0
	
	}
	
	Response protobuf message:
	
	{
	
	“output”: “0000000000000000000000000000000000000000000000000000000000000007”
	
	}

5. Light node service API

5.1. Get block information by hash

Endpoint	Blockchain explorer service
Method name	GetBlockInfoByHash
Method description	Extracting information from blockchain about requested block by block hash
Request parameters	[block_hash: bytes] Hash for requested block
Response parameters	[block_index: uint64] Block index in current blockchain [block_hash: bytes] Block hash [parent_hash: bytes] Block hash for previous block in block chain attached to this block [confirmation_timestamp: uint64] Time when block was confirmed in UNIX timestamp format [transaction_count: uint64] Number of transactions in block [transactions: Transaction array] All transactions for requested block
Response code descriptions	• OK – request successful • NOT_FOUND – block is not found

Custom message	[Transaction] - [sender_address: bytes] Address that initiated this transaction - [receiver_address: bytes] Address that received this transaction - [value: bytes] Amount send in transaction (for transfer fund) - [gas_limit: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) - [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) - [gas_used: bytes] Gas amount used executing transaction - [data: string] Smart contract bytecode in hex format - [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) - [new_address: bytes] New address that is created after executing transaction (deployed contract address) - [hash: bytes] Transaction hash - [block_index: uint64] Block index in current blockchain - [transaction_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format
Faulty scenarios	Block hash doesn’t exist in blockchain Block is not yet confirmed in blockchain

Example	Request protobuf message:
	
	{
	“block_hash”: “ c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”
	}
	
	Response protobuf message:
	“block_index”: 42,
	“block_hash”: “ c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”,
	“parent_hash”: “392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d”,
	“transaction_count": 2
	“confirmation_timestamp”: 1559653728

	“transactions”: [{
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	"value": 1000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 1
	"new_address": "0000000000000000000000000000000000000000000000000000"
	"hash": "392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d"
	"block_index": 42
	“transaction_timestamp”: 1559653728},
	{
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	"value": 2000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 2
	"new_address": "0000000000000000000000000000000000000000000000000000"
	"hash": "1223323ab2231a249b5b223ae2122ac0671b91902e2b39294fbe1c4f0ed2f12a"
	"block_index": 42
	“transaction_timestamp”: 1559653739
	}
	]}

5.2. Get block information by index

Endpoint	Blockchain explorer service
Method name	GetBlockInfoByIndex
Method description	Extracting information from blockchain about requested block by block index
Request parameters	[block_index: uint64] Index for requested block

Response parameters	[block_index: uint64] Block index in current blockchain [block_hash: bytes] Block hash [parent_hash: bytes] Block hash for previous block in block chain attached to this block [confirmation_timestamp: uint64] Time when block was confirmed in UNIX timestamp format [transaction_count: uint64] Number of transactions in block [transactions: Transaction array] All transactions for requested block
Response code descriptions	• OK – request successful • NOT_FOUND – block is not found
Custom message	[Transaction] - [sender_address: bytes] Address that initiated this transaction - [receiver_address: bytes] Address that received this transaction - [value: bytes] Amount send in transaction (for transfer fund) - [gas_limit: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) - [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) - [gas_used: bytes] Gas amount used executing transaction - [data: string] Smart contract bytecode in hex format - [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) - [new_address: bytes] New address that is created after executing transaction (deployed contract address) - [hash: bytes] Transaction hash - [block_index: uint64] Block index in current blockchain - [transaction_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format

Faulty scenarios	Block index doesn’t exist in blockchain Block is not yet confirmed in blockchain

Example

Request protobuf message:

{

“block_index”: 42

}

Response protobuf message:

“block_index”: 42,

“block_hash”: “ c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”,

“parent_hash”: “392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d”,

“transaction_count": 2

“confirmation_timestamp”: 1559653728

“transactions”: [{

“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,

“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,

"value": 1000,

"gas_limit": 21000,

"gas_price": 1,

"gas_used": 21000,

"data": "",

"nonce": 1

"new_address": "00000000000000000000000000000000000000000000000000"

"hash": "392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d"

"block_index": 42

“transaction_timestamp”: 1559653728},

{

“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,

“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,

"value": 2000,

"gas_limit": 21000,

"gas_price": 1,

"gas_used": 21000,

"data": "",

"nonce": 2

"new_address": "00000000000000000000000000000000000000000000000000"

"hash": "1223323ab2231a249b5b223ae2122ac0671b91902e2b39294fbe1c4f0ed2f12a"

"block_index": 42

“transaction_timestamp”: 1559653739

}

]}

5.3. Get latest blocks

Endpoint	Blockchain explorer service
Method name	GetLatestBlocks
Method description	Extracting information about lasest requested number of blocks from blockchain
Request parameters	[blocks_count: uint64] Number of latest blocks requested
Response parameters	[blocks: Blocks array] Requested number of latest blocks
Response code descriptions	• OK – request successful • NOT_FOUND – blockchain is empty

Custom message	[Block] - [block_index: uint64] Block index in current blockchain - [block_hash: bytes] Block hash - [parent_hash: bytes] Block hash for previous block in block chain attached to this block - [confirmation_timestamp: uint64] Time when block was confirmed in UNIX timestamp format - [transaction_count: uint64] Number of transactions in block - [transactions: Transaction array] All transactions for requested block [Transaction] - [sender_address: bytes] Address that initiated this transaction - [receiver_address: bytes] Address that received this transaction - [value: bytes] Amount send in transaction (for transfer fund) - [gas_limit: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) - [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) - [gas_used: bytes] Gas amount used executing transaction - [data: string] Smart contract bytecode in hex format - [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) - [new_address: bytes] New address that is created after executing transaction (deployed contract address) - [hash: bytes] Transaction hash - [block_index: uint64] Block index in current blockchain - [transaction_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format

Faulty scenarios	Cant find any blocks in blockchain

Example	Request protobuf message:
	
	{
	“blocks_count”: 1
	}
	
	Response protobuf message:
	"blocks": [{
	“block_index”: 42,
	“block_hash”: “ c5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470”,
	“parent_hash”: “392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d”,
	“transaction_count": 2
	“confirmation_timestamp”: 1559653728
	“transactions”: [{
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	"value": 1000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 1
	"new_address": "00000000000000000000000000000000000000000000000000"
	"hash": "392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d"
	"block_index": 42
	“transaction_timestamp”: 1559653728},
	{
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	“receiver_address": “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	"value": 2000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 2
	"new_address": "00000000000000000000000000000000000000000000000000"
	"hash": "1223323ab2231a249b5b223ae2122ac0671b91902e2b39294fbe1c4f0ed2f12a"
	"block_index": 42
	“transaction_timestamp”: 1559653739

	}]}
	
	]}

5.4. Get transaction by hash

Endpoint	Blockchain explorer service
Method name	GetTransaction
Method description	Extracting transaction information from blockchain
Request parameters	[transaction_hash: bytes] Transaction hash
Response parameters	[transaction: Transaction] Transaction
Response code descriptions	• OK – request successful • NOT_FOUND – transaction is not found
Custom message	[Transaction] - [sender_address: bytes] Address that initiated this transaction - [receiver_address: bytes] Address that received this transaction - [value: bytes] Amount send in transaction (for transfer fund) - [gas_limit: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) - [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) - [gas_used: bytes] Gas amount used executing transaction - [data: string] Smart contract bytecode in hex format - [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) - [new_address: bytes] New address that is created after executing transaction (deployed contract address) - [hash: bytes] Transaction hash - [block_index: uint64] Block index in current blockchain - [transaction_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format

Faulty scenarios	Transaction hash doesn't exist in blockchain Transaction is not yet confirmed in blockchain

Example	Request protobuf message:
	
	{
	"transaction_hash": "392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d"
	}
	
	Response protobuf message:
	"transaction": {
	"sender_address": "54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642",
	"receiver_address": "54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642",
	"value": 1000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 1
	"new_address": "00000000000000000000000000000000000000000000000000"
	"hash": "392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d"
	"block_index": 42
	"transaction_timestamp": 1559653728}
	}

5.5. Get address information

Endpoint	Blockchain explorer service
Method name	GetAddress
Method description	Extracting address information from blockchain
Request parameters	[address: bytes] Address
Response parameters	[balance: bytes] Address balance [transactions: Transaction array] All transactions for requested address
Response code descriptions	• OK – request successful • NOT_FOUND – address is not found

Custom message	[Transaction] - [sender_address: bytes] Address that initiated this transaction - [receiver_address: bytes] Address that received this transaction - [value: bytes] Amount send in transaction (for transfer fund) - [gas_limit: bytes] Maximum gas (gas limit) to spend to send this transaction (gas used for transaction sending or computational work in case of smart contracts) - [gas_price: bytes] Amount of gas to pay for each unit of gas, greater gas price is related to faster time to execute transaction (transaction fee = gas * gas price) - [gas_used: bytes] Gas amount used executing transaction - [data: string] Smart contract bytecode in hex format - [nonce: bytes] Unique transaction index for this sender address (auto-incremented value, each transaction has unique nonce) - [new_address: bytes] New address that is created after executing transaction (deployed contract address) - [hash: bytes] Transaction hash - [block_index: uint64] Block index in current blockchain - [transaction_timestamp: uint64] Time when transaction was confirmed in UNIX timestamp format
Faulty scenarios	Address didn't appear in any blockchain transactions. In that case present balance as 0 with message that there is no transactions in blockchain

Example	Request protobuf message:
	
	{
	“address”: "54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642"
	}
	
	Response protobuf message:
	“balance”: 1000000000,
	“transactions”: [{
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	“receiver_address": “654321416d894487e7bbd9d74065f7996cbdbf52bab123456”,
	"value": 1000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 1
	"new_address": "00000000000000000000000000000000000000000000000000"
	"hash": "392c3138b6931a649b5b293ae2129ac0671b91905ebb39694fbe1c4f0ed9f28d"
	"block_index": 42
	“transaction_timestamp”: 1559653728},
	{
	“sender_address”: “54f9f02416d894487e7bbd9d74065f7996cbdbf52bab547642”,
	“receiver_address": “1234f02416d894487e70000d74065f79962222f52bab332211”,
	"value": 2000,
	"gas_limit": 21000,
	"gas_price": 1,
	"gas_used": 21000,
	"data": "",
	"nonce": 2
	"new_address": "00000000000000000000000000000000000000000000000000"
	"hash": "1223323ab2231a249b5b223ae2122ac0671b91902e2b39294fbe1c4f0ed2f12a"
	"block_index": 24
	“transaction_timestamp”: 1559653739
	}
	]}