## Cypherium BlockChain JSON-RPC methods

-------------------------------------------------------------------------------------------

### net\_version

Returns the ID of the currently connected network

| **Client** | **Method invocation** |
| --- | --- |
| Console | net.version |
| RPC | {"method": "net\_version", "params": []} |

### net\_peerCount

Returns the number of peer nodes currently connected to the client.

| **Client** | **Method invocation** |
| --- | --- |
| Console | net.peerCount |
| RPC | {"method": "net\_peerCount", "params": []} |

### miner\_start

Start the CPU mining process with the given number of threads and generate a new DAG if need be.

| **Client** | **Method invocation** |
| --- | --- |
| Console | miner.start(number, address, password) |
| RPC | {"method": "miner\_start", "params": [number,address]} |

### miner\_stop

Stop the CPU mining operation.

| **Client** | **Method invocation** |
| --- | --- |
| Console | miner.stop() |
| RPC | {"method": "miner\_stop", "params": []} |

### miner\_content

the current status of all candidate nodes excavated to the mine.

| **Client** | **Method invocation** |
| --- | --- |
| Console | miner.content() |
| RPC | {"method": "miner\_content", "params": []} |

### miner\_status

the mining status of the current node and whether it is a leader.

| **Client** | **Method invocation** |
| --- | --- |
| Console | miner.status() |
| RPC | {"method": "miner\_status", "params": []} |

### admin\_addPeer

The addPeer administrative method requests adding a new remote node to the list of tracked static nodes. The node will try to maintain connectivity to these nodes at all times, reconnecting every once in a while if the remote connection goes down.

The method accepts a single argument, the [enode](https://github.com/ethereum/wiki/wiki/enode-url-format) URL of the remote peer to start tracking and returns a BOOL indicating whether the peer was accepted for tracking or some error occurred.

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.addPeer(url) |
| RPC | {"method": "admin\_addPeer", "params": ["enode://a979fb575495b8d6db44f750317d0f4622bf4c2aa3365d6af7c284339968eef29b69ad0dce72a4d8db5ebb4968de0e3bec910127f134779fbcb0cb6d3331163c@52.16.188.185:8000"]} |

### admin\_datadir

The datadir administrative property can be queried for the absolute path the running Geth node currently uses to store all its databases.

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.datadir |
| RPC | {"method": "admin\_datadir"} |

### admin\_nodeInfo

The nodeInfo administrative property can be queried for all the information known about the running Geth node at the networking granularity. These include general information about the node itself as a participant of the  P2P overlay protocol, as well as specialized information added by each of the running application protocols (e.g.cph, shh, bzz).

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.nodeInfo() |
| RPC | {"method": "admin\_nodeInfo"} |

### admin\_peers

The peers administrative property can be queried for all the information known about the connected remote nodes at the networking granularity. These include general information about the nodes themselves as participants of the  P2P overlay protocol, as well as specialized information added by each of the running application protocols (e.g.cph, shh, bzz).

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.peers |
| RPC | {"method": "admin\_peers"} |

### admin\_startRPC

The startRPC administrative method starts an HTTP based [JSON RPC](http://www.jsonrpc.org/specification) API webserver to handle client requests. All the parameters are optional:

* host: network interface to open the listener socket on (defaults to "localhost")
* port: network port to open the listener socket on (defaults to 8545)
* cors: [cross-origin resource sharing](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) header to use (defaults to "")
* apis: API modules to offer over this interface (defaults to "cph,net,web3")

The method returns a boolean flag specifying whether the HTTP RPC listener was opened or not. Please note, only one HTTP endpoint is allowed to be active at any time.

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.startRPC(host, port, cors, apis) |
| RPC | {"method": "admin\_startRPC", "params": [host, port, cors, apis]} |

### admin\_startWS

The startWS administrative method starts an WebSocket based [JSON RPC](http://www.jsonrpc.org/specification) API webserver to handle client requests. All the parameters are optional:

* host: network interface to open the listener socket on (defaults to "localhost")
* port: network port to open the listener socket on (defaults to 8000)
* cors: [cross-origin resource sharing](https://en.wikipedia.org/wiki/Cross-origin_resource_sharing) header to use (defaults to "")
* apis: API modules to offer over this interface (defaults to "cph,net,web3")

The method returns a boolean flag specifying whether the WebSocket RPC listener was opened or not. Please note, only one WebSocket endpoint is allowed to be active at any time.

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.startWS(host, port, cors, apis) |
| RPC | {"method": "admin\_startWS", "params": [host, port, cors, apis]} |

#### Example

> admin.startWS("127.0.0.1", 8000)true

### admin\_stopRPC

The stopRPC administrative method closes the currently open HTTP RPC endpoint. As the node can only have a single HTTP endpoint running, this method takes no parameters, returning a boolean whether the endpoint was closed or not.

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.stopRPC() |
| RPC | {"method": "admin\_stopRPC" |

### admin\_stopWS

The stopWS administrative method closes the currently open WebSocket RPC endpoint. As the node can only have a single WebSocket endpoint running, this method takes no parameters, returning a boolean whether the endpoint was closed or not.

| **Client** | **Method invocation** |
| --- | --- |
| Console | admin.stopWS() |
| RPC | {"method": "admin\_stopWS" |

### personal\_importRawKey

Imports the given unencrypted private key (hex string) into the key store, encrypting it with the passphrase.

Returns the address of the new account.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.importRawKey(keydata, passphrase) |
| RPC | {"method": "personal\_importRawKey", "params": [string, string]} |

### personal\_listAccounts

Returns all the Cypherium account addresses of all keys in the key store.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.listAccounts |
| RPC | {"method": "personal\_listAccounts", "params": []} |

### personal\_lockAccount

Removes the private key with given address from memory. The account can no longer be used to send transactions.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.lockAccount(address) |
| RPC | {"method": "personal\_lockAccount", "params": [string]} |

### personal\_newAccount

Generates a new private key and stores it in the key store directory. The key file is encrypted with the given passphrase. Returns the address of the new account.

At the geth console, newAccount will prompt for a passphrase when it is not supplied as the argument.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.newAccount() |
| RPC | {"method": "personal\_newAccount", "params": [string]} |

### personal\_unlockAccount

Decrypts the key with the given address from the key store.

Both passphrase and unlock duration are optional when using the JavaScript console. If the passphrase is not supplied as an argument, the console will prompt for the passphrase interactively.

The unencrypted key will be held in memory until the unlock duration expires. If the unlock duration defaults to 300 seconds. An explicit duration of zero seconds unlocks the key until geth exits.

The account can be used with cph\_sign and cph\_sendTransaction while it is unlocked.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.unlockAccount(address, passphrase, duration) |
| RPC | {"method": "personal\_unlockAccount", "params": [string, string, number]} |

#### Examples

> personal.unlockAccount("0x5e97870f263700f46aa00d967821199b9bc5a120")Unlock account 0x5e97870f263700f46aa00d967821199b9bc5a120Passphrase: true

Supplying the passphrase and unlock duration as arguments:

> personal.unlockAccount("0x5e97870f263700f46aa00d967821199b9bc5a120", "foo", 30)true

If you want to type in the passphrase and stil override the default unlock duration, pass null as the passphrase.

> personal.unlockAccount("0x5e97870f263700f46aa00d967821199b9bc5a120", null, 30)

Unlock account 0x5e97870f263700f46aa00d967821199b9bc5a120

Passphrase:

true

### personal\_sendTransaction

Validate the given passphrase and submit transaction.

The transaction is the same argument as for cph\_sendTransaction and contains the from address. If the passphrase can be used to decrypt the private key belogging to tx.from the transaction is verified, signed and send onto the network. The account is not unlocked globally in the node and cannot be used in other RPC calls.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.sendTransaction(tx, passphrase) |
| RPC | {"method": "personal\_sendTransaction", "params": [tx, string]} |

Note, prior to Geth 1.5, please use personal\_signAndSendTransaction as that was the original introductory name and only later renamed to the current final version.

#### Examples

> var tx = {from: "0x391694e7e0b0cce554cb130d723a9d27458f9298", to: "0xafa3f8684e54059998bc3a7b0d2b0da075154d66", value: web3.toWei(1.23, "ether")}undefined> personal.sendTransaction(tx, "passphrase")0x8474441674cdd47b35b875fd1a530b800b51a5264b9975fb21129eeb8c18582f

### personal\_sign

The sign method calculates an Cypherium specific signature with: sign(keccack256("\x19Cypherium Signed Message:\n" + len(message) + message))).

By adding a prefix to the message makes the calculated signature recognisable as an Cypherium specific signature. This prevents misuse where a malicious DApp can sign arbitrary data (e.g. transaction) and use the signature to impersonate the victim.

See ecRecover to verify the signature.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.sign(message, account, [password]) |
| RPC | {"method": "personal\_sign", "params": [message, account, password]} |

#### Examples

> personal.sign("0xdeadbeaf", "0x9b2055d370f73ec7d8a03e965129118dc8f5bf83", "")"0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9f709d39ce8325427b57afcfc994cee1b"

### personal\_ecRecover

ecRecover returns the address associated with the private key that was used to calculate the signature in personal\_sign.

| **Client** | **Method invocation** |
| --- | --- |
| Console | personal.ecRecover(message, signature) |
| RPC | {"method": "personal\_ecRecover", "params": [message, signature]} |

#### Examples

> personal.sign("0xdeadbeaf", "0x9b2055d370f73ec7d8a03e965129118dc8f5bf83", "")"0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9f709d39ce8325427b57afcfc994cee1b"> personal.ecRecover("0xdeadbeaf", "0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9f709d39ce8325427b57afcfc994cee1b")"0x9b2055d370f73ec7d8a03e965129118dc8f5bf83"

### txpool\_content

The content inspection property can be queried to list the exact details of all the transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future execution only.

The result is an object with two fields pending and queued. Each of these fields are associative arrays, in which each entry maps an origin-address to a batch of scheduled transactions. These batches themselves are maps associating nonces with actual transactions.

Please note, there may be multiple transactions associated with the same account and nonce. This can happen if the user broadcast mutliple ones with varying gas allowances (or even complerely different transactions).

| **Client** | **Method invocation** |
| --- | --- |
| Console | txpool.content |
| RPC | {"method": "txpool\_content"} |

#### Example

> txpool.content{

pending: {

0x0216d5032f356960cd3749c31ab34eeff21b3395: {

806: {

blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",

blockNumber: null,

from: "0x0216d5032f356960cd3749c31ab34eeff21b3395",

gas: "0x5208",

gasPrice: "0xba43b7400",

hash: "0xaf953a2d01f55cfe080c0c94150a60105e8ac3d51153058a1f03dd239dd08586",

input: "0x",

nonce: "0x326",

to: "0x7f69a91a3cf4be60020fb58b893b7cbb65376db8",

transactionIndex: null,

value: "0x19a99f0cf456000"

}

},

0x24d407e5a0b506e1cb2fae163100b5de01f5193c: {

34: {

blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",

blockNumber: null,

from: "0x24d407e5a0b506e1cb2fae163100b5de01f5193c",

gas: "0x44c72",

gasPrice: "0x4a817c800",

hash: "0xb5b8b853af32226755a65ba0602f7ed0e8be2211516153b75e9ed640a7d359fe",

input: "0xb61d27f600000000000000000000000024d407e5a0b506e1cb2fae163100b5de01f5193c00000000000000000000000000000000000000000000000053444835ec580000000000000000000000000000000000000000000000000000000000000000006000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",

nonce: "0x22",

to: "0x7320785200f74861b69c49e4ab32399a71b34f1a",

transactionIndex: null,

value: "0x0"

}

}

},

queued: {

0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c: {

3: {

blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",

blockNumber: null,

from: "0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c",

gas: "0x15f90",

gasPrice: "0x4a817c800",

hash: "0x57b30c59fc39a50e1cba90e3099286dfa5aaf60294a629240b5bbec6e2e66576",

input: "0x",

nonce: "0x3",

to: "0x346fb27de7e7370008f5da379f74dd49f5f2f80f",

transactionIndex: null,

value: "0x1f161421c8e0000"

}

},

0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a: {

2: {

blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",

blockNumber: null,

from: "0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a",

gas: "0x15f90",

gasPrice: "0xba43b7400",

hash: "0x3a3c0698552eec2455ed3190eac3996feccc806970a4a056106deaf6ceb1e5e3",

input: "0x",

nonce: "0x2",

to: "0x24a461f25ee6a318bdef7f33de634a67bb67ac9d",

transactionIndex: null,

value: "0xebec21ee1da40000"

},

6: {

blockHash: "0x0000000000000000000000000000000000000000000000000000000000000000",

blockNumber: null,

from: "0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a",

gas: "0x15f90",

gasPrice: "0x4a817c800",

hash: "0xbbcd1e45eae3b859203a04be7d6e1d7b03b222ec1d66dfcc8011dd39794b147e",

input: "0x",

nonce: "0x6",

to: "0x6368f3f8c2b42435d6c136757382e4a59436a681",

transactionIndex: null,

value: "0xf9a951af55470000"

}

}

}}

### txpool\_inspect

The inspect inspection property can be queried to list a textual summary of all the transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future execution only. This is a method specifically tailored to developers to quickly see the transactions in the pool and find any potential issues.

The result is an object with two fields pending and queued. Each of these fields are associative arrays, in which each entry maps an origin-address to a batch of scheduled transactions. These batches themselves are maps associating nonces with transactions summary strings.

Please note, there may be multiple transactions associated with the same account and nonce. This can happen if the user broadcast mutliple ones with varying gas allowances (or even complerely different transactions).

| **Client** | **Method invocation** |
| --- | --- |
| Console | txpool.inspect |
| RPC | {"method": "txpool\_inspect"} |

#### Example

> txpool.inspect{

pending: {

0x26588a9301b0428d95e6fc3a5024fce8bec12d51: {

31813: "0x3375ee30428b2a71c428afa5e89e427905f95f7e: 0 wei + 500000 × 20000000000 wei"

},

0x2a65aca4d5fc5b5c859090a6c34d164135398226: {

563662: "0x958c1fa64b34db746925c6f8a3dd81128e40355e: 1051546810000000000 wei + 90000 gas × 20000000000 wei",

563663: "0x77517b1491a0299a44d668473411676f94e97e34: 1051190740000000000 wei + 90000 gas × 20000000000 wei",

563664: "0x3e2a7fe169c8f8eee251bb00d9fb6d304ce07d3a: 1050828950000000000 wei + 90000 gas × 20000000000 wei",

563665: "0xaf6c4695da477f8c663ea2d8b768ad82cb6a8522: 1050544770000000000 wei + 90000 gas × 20000000000 wei",

563666: "0x139b148094c50f4d20b01caf21b85edb711574db: 1048598530000000000 wei + 90000 gas × 20000000000 wei",

563667: "0x48b3bd66770b0d1eecefce090dafee36257538ae: 1048367260000000000 wei + 90000 gas × 20000000000 wei",

563668: "0x468569500925d53e06dd0993014ad166fd7dd381: 1048126690000000000 wei + 90000 gas × 20000000000 wei",

563669: "0x3dcb4c90477a4b8ff7190b79b524773cbe3be661: 1047965690000000000 wei + 90000 gas × 20000000000 wei",

563670: "0x6dfef5bc94b031407ffe71ae8076ca0fbf190963: 1047859050000000000 wei + 90000 gas × 20000000000 wei"

},

0x9174e688d7de157c5c0583df424eaab2676ac162: {

3: "0xbb9bc244d798123fde783fcc1c72d3bb8c189413: 30000000000000000000 wei + 85000 gas × 21000000000 wei"

},

0xb18f9d01323e150096650ab989cfecd39d757aec: {

777: "0xcd79c72690750f079ae6ab6ccd7e7aedc03c7720: 0 wei + 1000000 gas × 20000000000 wei"

},

0xb2916c870cf66967b6510b76c07e9d13a5d23514: {

2: "0x576f25199d60982a8f31a8dff4da8acb982e6aba: 26000000000000000000 wei + 90000 gas × 20000000000 wei"

},

0xbc0ca4f217e052753614d6b019948824d0d8688b: {

0: "0x2910543af39aba0cd09dbb2d50200b3e800a63d2: 1000000000000000000 wei + 50000 gas × 1171602790622 wei"

},

0xea674fdde714fd979de3edf0f56aa9716b898ec8: {

70148: "0xe39c55ead9f997f7fa20ebe40fb4649943d7db66: 1000767667434026200 wei + 90000 gas × 20000000000 wei"

}

},

queued: {

0x0f6000de1578619320aba5e392706b131fb1de6f: {

6: "0x8383534d0bcd0186d326c993031311c0ac0d9b2d: 9000000000000000000 wei + 21000 gas × 20000000000 wei"

},

0x5b30608c678e1ac464a8994c3b33e5cdf3497112: {

6: "0x9773547e27f8303c87089dc42d9288aa2b9d8f06: 50000000000000000000 wei + 90000 gas × 50000000000 wei"

},

0x976a3fc5d6f7d259ebfb4cc2ae75115475e9867c: {

3: "0x346fb27de7e7370008f5da379f74dd49f5f2f80f: 140000000000000000 wei + 90000 gas × 20000000000 wei"

},

0x9b11bf0459b0c4b2f87f8cebca4cfc26f294b63a: {

2: "0x24a461f25ee6a318bdef7f33de634a67bb67ac9d: 17000000000000000000 wei + 90000 gas × 50000000000 wei",

6: "0x6368f3f8c2b42435d6c136757382e4a59436a681: 17990000000000000000 wei + 90000 gas × 20000000000 wei",

7: "0x6368f3f8c2b42435d6c136757382e4a59436a681: 17900000000000000000 wei + 90000 gas × 20000000000 wei"

}

}}

### txpool\_status

The status inspection property can be queried for the number of transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future execution only.

The result is an object with two fields pending and queued, each of which is a counter representing the number of transactions in that particular state.

| **Client** | **Method invocation** |
| --- | --- |
| Console | txpool.status |
| RPC | {"method": "txpool\_status"} |

#### Example

> txpool.status{

pending: 10,

queued: 7}

### cph\_syncing

Returns an object with data about the sync status or false.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.syncing |
| RPC | {"method": "cph\_syncing", "params": []} |

### cph\_coinbase

Returns the client coinbase address.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.coinbase |
| RPC | {"method": "cph\_coinbase", "params": []} |

### cph\_hashrate

Returns the number of hashes per second that the node is mining with.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.hashrete |
| RPC | {"method": "cph\_hashrate", "params": []} |

### cph\_accounts

Returns a list of addresses owned by client.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.accounts |
| RPC | {"method": "cph\_accounts", "params": []} |

### cph\_txBlockNumber

Returns the number of most recent block.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.txBlockNumber |
| RPC | {"method": "cph\_txBlockNumber", "params": []} |

### cph\_keyBlockNumber

Returns the number of most recent key block.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.keyBlockNumber |
| RPC | {"method": "cph\_keyBlockNumber", "params": []} |

### cph\_gasPrice

Returns the current price per gas.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.gasPrice |
| RPC | {"method": "cph\_gasPrice", "params": []} |

### cph\_getBalance

Returns the balance of the account of given address.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getBalance(address) |
| RPC | {"method": "cph\_getBalance", "params": [‘0x9b2055d370f73ec7d8a03e965129118dc8f5bf83’]} |

### cph\_getStorageAt

##### Returns the value from a storage position at a given address. Parameters: 1.address address of the storage. 2.position integer of the position in the storage. 3.tag integer block number, or the string "latest", "earliest" or "pending"

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getStorageAt(address,position,tag) |
| RPC | {"method": "cph\_getStorageAt", "params": ['0x407d73d8a49eeb85d32cf465507dd71d507100c1',  '0x0', // storage position at 0  '0x2' // state at block number 2]} |

### cph\_getTransactionCount

Returns the number of transactions send from a address.  
Parameters  
1.address,

2.TAG - integer block number, or the string "latest", "earliest" or "pending".

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTransactionCount(address,tag) |
| RPC | {"method": "cph\_getTransactionCount", "params": ['0x407d73d8a49eeb85d32cf465507dd71d507100c1',  'latest' // state at the latest block]} |

### cph\_getBlockTransactionCountByHash

Returns the number of transactions in a block from a block matching the given block hash.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getBlockTransactionCountByHash(hash) |
| RPC | {"method": "cph\_getBlockTransactionCountByHash", "params": ['0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238']} |

### cph\_getBlockTransactionCountByNumber

Returns the number of transactions in a block matching the given block number.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getBlockTransactionCountByNumber(number) |
| RPC | {"method": "cph\_getBlockTransactionCountByNumber", "params": ['0xe8']} |

### cph\_getCode

Returns the number of transactions in a block matching the given block number.  
Parameters

1. address,

2.TAG - integer block number, or the string "latest", "earliest" or "pending"

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getCode(address,tag) |
| RPC | {"method": "cph\_getCode", "params": ['0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b','0x2' // 2]} |

### cph\_sendTransaction

Creates new message call transaction or a contract creation, if the data field contains code.  
Parameters

1.Object - The transaction object

from: - The address the transaction is send from.

to: - (optional when creating new contract) The address the transaction is directed to.

gas: - (optional, default: To-Be-Determined) Integer of the gas provided for the transaction execution. It will return unused gas.

gasPrice: - (optional, default: To-Be-Determined) Integer of the gasPrice used for each payed gas

value: - (optional) Integer of the value send with this transaction

data: - (optional) The compiled code of a contract

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.sendTransaction(obj) |
| RPC | {"method": "cph\_sendTransaction", "params": [{  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",  "to": "0xd46e8dd67c5d32be8058bb8eb970870f072445675",  "gas": "0x76c0", // 30400,  "gasPrice": "0x9184e72a000", // 10000000000000  "value": "0x9184e72a", // 2441406250  "data": "0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675"  }]} |

### cph\_sendRawTransaction

Creates new message call transaction or a contract creation for signed transactions.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.sendRawTransaction(data) |
| RPC | {"method": "cph\_sendRawTransaction", "params": [...]} |

### cph\_getTransactionReceipt

Returns the receipt of a transaction by transaction hash.  
Note That the receipt is not available for pending transactions.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTransactionReceipt(hash) |
| RPC | {"method": "cph\_getTransactionReceipt", "params": ['0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238']} |

### cph\_call

Executes a new message call immediately without creating a transaction on the block chain.  
Parameters

1.Object - The transaction call object

from: - The address the transaction is send from.

to: - The address the transaction is directed to.

gas: - (optional) Integer of the gas provided for the transaction execution. It will return unused gas.

gasPrice: - (optional) Integer of the gasPrice used for each payed gas

value: - (optional) Integer of the value send with this transaction

data: - (optional) The compiled code of a contract

2.TAG - integer block number, or the string "latest", "earliest" or "pending"

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.call(obj,tag) |
| RPC | {"method": "cph\_call", "params": [...]} |

### cph\_getTxBlockByHash

Returns information about a block by hash.

Parameters:  
1.hash

1. inclTx If inclTx is true the returned block contains transactions hash.
2. fullTx if fullTx is true the returned block contains full transaction details

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTxBlockByHash(hash,boolean) |
| RPC | {"method": "cph\_getTxBlockByHash", "params": [‘0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331‘, true, false]} |

### cph\_getTxBlockByNumber

Returns information about a block by block number.  
Parameters:

1.block number

2.inclTx If inclTx is true the returned block contains transactions hash.

3.fullTx if fullTx is true the returned block contains full transaction details

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTxBlockByNumber(number,boolean) |
| RPC | {"method": "cph\_getTxBlockByNumber", "params": ['0x1b4',true, false]} |

### cph\_getKeyBlockByHash

Returns information about a keyblock by hash.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getKeyBlockByHash(hash) |
| RPC | {"method": "cph\_getKeyBlockByHash", "params": [‘0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331‘]} |

### cph\_getKeyBlockByNumber

Returns information about a key block by block number.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getKeyBlockByNumber(number) |
| RPC | {"method": "cph\_getKeyBlockByNumber", "params": ['0x1b4']} |

### cph\_getTransactionByHash

Returns the information about a transaction requested by transaction hash.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTransactionByHash(hash) |
| RPC | {"method": "cph\_getTransactionByHash", "params": ["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"]} |

### cph\_getTransactionByBlockHashAndIndex

Returns information about a transaction by block hash and transaction index position.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTransactionByBlockHashAndIndex(hash,index) |
| RPC | {"method": "cph\_getTransactionByBlockHashAndIndex", "params": ['0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331','0x0']} |

### cph\_getTransactionByBlockNumberAndIndex

Returns information about a transaction by block number and transaction index position.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getTransactionByBlockNumberAndIndex(number,index) |
| RPC | {"method": "cph\_getTransactionByBlockNumberAndIndex", "params": ['0x29c',’0x0’]} |

### cph\_sign

The sign method calculates an specific signature

Note the address to sign with must be unlocked.  
Parameters

1.address

2.message message to sign

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.sign(address,message) |
| RPC | {"method": "cph\_sign", "params": ["0x9b2055d370f73ec7d8a03e965129118dc8f5bf83", "0xdeadbeaf"]} |

### cph\_signTransaction

Signs a transaction that can be submitted to the network at a later time using with cph\_sendRawTransaction

Parameters

1.Object - The transaction object

from: - The address the transaction is sent from.

to: - (optional when creating new contract) The address the transaction is directed to.

gas: - (optional, default: 90000) Integer of the gas provided for the transaction execution. It will return unused gas.

gasPrice: - (optional, default: To-Be-Determined) Integer of the gasPrice used for each paid gas, in Wei.

value: - (optional) Integer of the value sent with this transaction, in Wei.

data: - The compiled code of a contract OR the hash of the invoked method signature and encoded parameters. For details see Cypherium Contract ABI.

nonce: - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.signTransaction(obj) |
| RPC | {"method": "cph\_signTransaction", "params": [{  "data": "0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675",  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",  "gas": "0x76c0",  "gasPrice": "0x9184e72a000",  "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",  "value": "0x9184e72a"  }]} |

### cph\_estimateGas

Generates and returns an estimate of how much gas is necessary to allow the transaction to complete. The transaction will not be added to the blockchain. Note that the estimate may be significantly more than the amount of gas actually used by the transaction, for a variety of reasons including EVM mechanics and node performance.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.estimateGas(data) |
| RPC | {"method": "cph\_estimateGas", "params": [...]} |

### cph\_compileSolidity

Returns compiled solidity code.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.compileSolidity(string) |
| RPC | {"method": "cph\_compileSolidity", "params": ["contract test { function multiply(uint a) returns(uint d){ return a \* 7;} }"]} |

### cph\_newFilter

Creates a filter object, based on filter options, to notify when the state changes (logs).

To check if the state has changed, call cph\_getFilterChanges.

A note on specifying topic filters:

Topics are order-dependent. A transaction with a log with topics [A, B] will be matched by the following topic filters:

[ ] “anything”

[A] “A in first position (and anything after)”

[null, B] “anything in first position AND B in second position (and anything after)”

[A, B] “A in first position AND B in second position (and anything after)”

[[A, B], [A, B]] “(A OR B) in first position AND (A OR B) in second position (and anything after)”.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.newFilter(obj)  Object - The filter options:  fromBlock: QUANTITY|TAG - (optional, default: "latest") Integer block number, or "latest" for the last mined block or "pending", "earliest" for not yet mined transactions.  toBlock: QUANTITY|TAG - (optional, default: "latest") Integer block number, or "latest" for the last mined block or "pending", "earliest" for not yet mined transactions.  address: DATA|Array, 20 Bytes - (optional) Contract address or a list of addresses from which logs should originate.  topics: Array of DATA, - (optional) Array of 32 Bytes DATA topics. |
| RPC | {"method": "cph\_newFilter", "params": [{  "fromBlock": "0x1",  "toBlock": "0x2",  "address": "0x8888f1f195afa192cfee860698584c030f4c9db1",  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]  }]} |

### cph\_newBlockFilter

Creates a filter in the node, to notify when a new block arrives.

To check if the state has changed, call cph\_getFilterChanges.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.newBlockFilter(tag) tag-- The string "latest" for notifications about new block and "pending" for notifications about pending transactions. |
| RPC | {"method": "cph\_newBlockFilter", "params": ["pending"]} |

### cph\_newPendingTransactionFilter

Creates a filter in the node, to notify when new pending transactions arrive.

To check if the state has changed, call cph\_getFilterChanges.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.newPendingTransactionFilter() |
| RPC | {"method": "cph\_newPendingTransactionFilter", "params": []} |

### cph\_uninstallFilter

Uninstalls a filter with given id. Should always be called when watch is no longer needed.

Additonally Filters timeout when they aren’t requested with cph\_getFilterChanges for a period of time.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.uninstallFilter(id) |
| RPC | {"method": "cph\_uninstallFilter", "params": ["0x16"]} |

### cph\_getFilterChanges

Polling method for a filter, which returns an array of logs which occurred since last poll.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getFilterChanges(id) id --- the filter id. |
| RPC | {"method": "cph\_getFilterChanges", "params": ["0x16"]} |

### cph\_getFilterLogs

Returns an array of all logs matching filter with given id.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getFilterLogs(id) id--The filter id |
| RPC | {"method": "cph\_getFilterLogs", "params": ["0x16"]} |

### cph\_getLogs

Returns an array of all logs matching a given filter object.

| **Client** | **Method invocation** |
| --- | --- |
| Console | cph.getLogs(obj)  obj --the filter object, see cph\_newFilter parameters. |
| RPC | {"method": "cph\_getLogs", "params": ["topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]]} |

Curl -H “Content-Type:application/json” -X POST --data ‘{"method": "cph\_getTxBlockByNumber", "params": ['0x1',true, false], "id":1}’ http://192.168.1.211:18002