RPC API DOCUMENTATION (JSON)

Syed Jaffar Raza

NOTE: This is my personal RPC API documentation, it is not an official version. It is to be used for the help of developers only.

RPC is a stateless, light-weight remote procedure call (RPC) protocol. Primarily this specification defines several data structures and the rules around their processing. It is transport agnostic in that the concepts can be used within the same process, over sockets, over HTTP, or in many various message passing environments. It uses JSON (RFC 4627) as data format.

Geth 1.4 has experimental pub/sub support. See this page for more information.

Parity 1.6 has experimental pub/sub support. See this for more information.

Pantheon 0.8 has pub/sub support. See this for more information.

JavaScript API

To talk to an ethereum node from inside a JavaScript application use the web3.js library, which gives a convenient interface for the RPC methods. See the JavaScript API for more.

JSON-RPC Endpoint

Default JSON-RPC endpoints:

Client	URL
C++	http://localhost:8545
Go	http://localhost:8545
Py	http://localhost:4000
Parity	http://localhost:8545
Pantheon	http://localhost:8545

Go

You can start the HTTP JSON-RPC with the `--rpc` flag

```
geth --rpc
```

change the default port (8545) and listing address (localhost) with:

```
geth --rpc --rpcaddr <ip> --rpcport <portnumber>
```

If accessing the RPC from a browser, CORS will need to be enabled with the appropriate domain set. Otherwise, JavaScript calls are limit by the same-origin policy and requests will fail:

```
geth --rpc --rpccorsdomain "http://localhost:3000"
```

The JSON RPC can also be started from the geth console using the `admin.startRPC(addr, port)` command.

C++

First start the node by running `aleth` application:

```
build/aleth/aleth
```

Then start the JSON-RPC proxy (defaults to '~/.ethereum/geth.ipc' and 'http://127.0.0.1:8545'):

```
scripts/jsonrpcproxy.py
```

If you use non-default IPC path or JSON-RPC options, you can specify :

```
scripts/jsonrpcproxy.py <path to your node's geth.ipc> <URL for this proxy server>
```

Python

In python the JSONRPC server is currently started by default and listens on `127.0.0.1:4000`
You can change the port and listen address by giving a config option.

```
pyethapp -c jsonrpc.listen_port=4002 -c jsonrpc.listen_host=127.0.0.2 run
```

JSON-RPC support

	cpp-ethereum	go-ethereum	py-ethereum	parity	pantheon
JSON-RPC 1.0	✓				

	cpp-ethereum	go-ethereum	py-ethereum	parity	pantheon
JSON-RPC 2.0	✓	✓	✓	✓	✓
Batch requests	✓	✓	✓	✓	✓
HTTP	✓	✓	✓	✓	✓
IPC	✓	✓		✓	
WS		✓		✓	✓

HEX value encoding

At present there are two key datatypes that are passed over JSON: unformatted byte arrays and quantities. Both are passed with a hex encoding, however with different requirements to formatting:

When encoding **QUANTITIES** (integers, numbers): encode as hex, prefix with "0x", the most compact representation (slight exception: zero should be represented as "0x0"). Examples:

- 0x41 (65 in decimal)
- 0x400 (1024 in decimal)
- WRONG: 0x (should always have at least one digit - zero is "0x0")
- WRONG: 0x0400 (no leading zeroes allowed)
- WRONG: ff (must be prefixed 0x)

When encoding **UNFORMATTED DATA** (byte arrays, account addresses, hashes, bytecode arrays): encode as hex, prefix with "0x", two hex digits per byte. Examples:

- 0x41 (size 1, "A")
- 0x004200 (size 3, "\0B\0")
- 0x (size 0, "")
- WRONG: 0xf0f0f (must be even number of digits)

- WRONG: 004200 (must be prefixed 0x)

Currently cpp-ethereum,go-ethereum and parity provide JSON-RPC communication over http and IPC (unix socket Linux and OSX/named pipes on Windows). Version 1.4 of go-ethereum, version 1.6 of Parity and version 0.8 of Pantheon onwards have websocket support.

The default block parameter

The following methods have an extra default block parameter:

- eth_getBalance
- eth_getCode
- eth_getTransactionCount
- eth_getStorageAt
- eth_call

When requests are made that act on the state of ethereum, the last default block parameter determines the height of the block.

The following options are possible for the defaultBlock parameter:

- `HEX String` - an integer block number
- `String "earliest"` for the earliest/genesis block
- `String "latest"` - for the latest mined block
- `String "pending"` - for the pending state/transactions

Curl Examples Explained

The curl options below might return a response where the node complains about the content type, this is because the --data option sets the content type to application/x-www-form-urlencoded . If your node does complain, manually set the header by placing -H "Content-Type: application/json" at the start of the call.

The examples also do not include the URL/IP & port combination which must be the last argument given to curl e.x. 127.0.0.1:8545

JSON-RPC methods

JSON RPC API Reference

---

web3_clientVersion

Returns the current client version.

Parameters

none

Returns

`String` - The current client version.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"web3_clientVersion","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc":"2.0",
  "result": "Mist/v0.9.3/darwin/go1.4.1"
}
```

---

web3_sha3

Returns Keccak-256 (*not* the standardized SHA3-256) of the given data.

Parameters

1. `DATA` - the data to convert into a SHA3 hash.

Example Parameters

```
params: [
  "0x68656c6c6f20776f726c64"
]
```

Returns

`DATA` - The SHA3 result of the given string.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"web3_sha3","params":["0x68656c6c6f20776f726c64"],"id":64}
'

// Result
{
  "id":64,
  "jsonrpc": "2.0",
  "result": "0x47173285a8d7341e5e972fc677286384f802f8ef42a5ec5f03bbfa254cb01fad"
}
```

---

net_version

Returns the current network id.

Parameters

none

Returns

`String` - The current network id.

- `"1"`: Ethereum Mainnet
- `"2"`: Morden Testnet (deprecated)
- `"3"`: Ropsten Testnet
- `"4"`: Rinkeby Testnet
- `"42"`: Kovan Testnet

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_version","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "3"
}
```

---

net_listening

Returns `true` if client is actively listening for network connections.

Parameters

none

Returns

`Boolean` - `true` when listening, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_listening","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc":"2.0",
  "result":true
}
```

---

net_peerCount

Returns number of peers currently connected to the client.

Parameters

none

Returns

`QUANTITY` - integer of the number of connected peers.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"net_peerCount","params":[],"id":74}'

// Result
{
  "id":74,
  "jsonrpc": "2.0",
  "result": "0x2" // 2
}
```

---

eth_protocolVersion

Returns the current ethereum protocol version.

Parameters

none

Returns

`String` - The current ethereum protocol version.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_protocolVersion","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "0x54"
}
```

---

eth_syncing

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

Parameters

none

Returns

`Object|Boolean`, An object with sync status data or `FALSE`, when not syncing:

- `startingBlock`: `QUANTITY` - The block at which the import started (will only be reset, after the sync reached his head)
- `currentBlock`: `QUANTITY` - The current block, same as eth_blockNumber
- `highestBlock`: `QUANTITY` - The estimated highest block

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_syncing","params":[],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
    startingBlock: '0x384',
    currentBlock: '0x386',
    highestBlock: '0x454'
  }
}
// Or when not syncing
{
  "id":1,
  "jsonrpc": "2.0",
  "result": false
}
```

---

eth_coinbase

Returns the client coinbase address.

Parameters

none

Returns

`DATA`, 20 bytes - the current coinbase address.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_coinbase","params":[],"id":64}'

// Result
{
  "id":64,
  "jsonrpc": "2.0",
  "result": "0xc94770007dda54cF92009BFF0dE90c06F603a09f"
}
```

---

eth_mining

Returns `true` if client is actively mining new blocks.

Parameters

none

Returns

`Boolean` - returns `true` of the client is mining, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_mining","params":[],"id":71}'

// Result
{
  "id":71,
  "jsonrpc": "2.0",
  "result": true
}
```

---

eth_hashrate

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

Parameters

none

Returns

`QUANTITY` - number of hashes per second.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_hashrate","params":[],"id":71}'

// Result
{
  "id":71,
  "jsonrpc": "2.0",
  "result": "0x38a"
}
```

---

eth_gasPrice

Returns the current price per gas in wei.

Parameters

none

Returns

`QUANTITY` - integer of the current gas price in wei.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_gasPrice","params":[],"id":73}'

// Result
{
  "id":73,
  "jsonrpc": "2.0",
  "result": "0x09184e72a000" // 10000000000000
}
```

---

eth_accounts

Returns a list of addresses owned by client.

Parameters

none

Returns

`Array of DATA`, 20 Bytes - addresses owned by the client.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_accounts","params":[],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": ["0xc94770007dda54cF92009BFF0dE90c06F603a09f"]
}
```

---

eth_blockNumber

Returns the number of most recent block.

Parameters

none

Returns

`QUANTITY` - integer of the current block number the client is on.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_blockNumber","params":[],"id":1}'

// Result
{
  "id":83,
  "jsonrpc": "2.0",
  "result": "0xc94" // 1207
}
```

---

eth_getBalance

Returns the balance of the account of given address.

Parameters

1. `DATA`, 20 Bytes - address to check for balance.
2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the default block parameter

Example Parameters

```
params: [
   '0xc94770007dda54cF92009BFF0dE90c06F603a09f',
   'latest'
]
```

Returns

`QUANTITY` - integer of the current balance in wei.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBalance","params":["0xc94770007dda54cF92009BFF0dE9
0c06F603a09f", "latest"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x0234c8a3397aab58" // 158972490234375000
}
```

---

eth_getStorageAt

Returns the value from a storage position at a given address.

Parameters

1. `DATA`, 20 Bytes - address of the storage.
2. `QUANTITY` - integer of the position in the storage.
3. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the default block parameter

Returns

`DATA` - the value at this storage position.

Example

Calculating the correct position depends on the storage to retrieve. Consider the following contract deployed at `0x295a70b2de5e3953354a6a8344e616ed314d7251` by address `0x391694e7e0b0cce554cb130d723a9d27458f9298`.

```
contract Storage {
    uint pos0;
    mapping(address => uint) pos1;

    function Storage() {
        pos0 = 1234;
        pos1[msg.sender] = 5678;
    }
}
```

Retrieving the value of pos0 is straight forward:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getStorageAt", "params":
["0x295a70b2de5e3953354a6a8344e616ed314d7251", "0x0", "latest"], "id": 1}'
localhost:8545

{"jsonrpc":"2.0","id":1,"result":"0x00000000000000000000000000000000000000000000000
00000000000004d2"}
```

Retrieving an element of the map is harder. The position of an element in the map is calculated with:

```
keccack(LeftPad32(key, 0), LeftPad32(map position, 0))
```

This means to retrieve the storage on pos1["0x391694e7e0b0cce554cb130d723a9d27458f9298"] we need to calculate the position with:

```
keccak(decodeHex("000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" +
"0000000000000000000000000000000000000000000000000000000000000001"))
```

The geth console which comes with the web3 library can be used to make the calculation:

```
> var key = "000000000000000000000000391694e7e0b0cce554cb130d723a9d27458f9298" +
"0000000000000000000000000000000000000000000000000000000000000001"
undefined
> web3.sha3(key, {"encoding": "hex"})
"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9"
```

Now to fetch the storage:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getStorageAt", "params":
["0x295a70b2de5e3953354a6a8344e616ed314d7251",
"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9", "latest"],
"id": 1}' localhost:8545

{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000
00000000000162e"}
```

---

eth_getTransactionCount

Returns the number of transactions *sent* from an address.

Parameters

1. `DATA`, 20 Bytes - address.
2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the default block parameter

Example Parameters

```
params: [
   '0xc94770007dda54cF92009BFF0dE90c06F603a09f',
   'latest' // state at the latest block
]
```

Returns

`QUANTITY` - integer of the number of transactions send from this address.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionCount","params":["0xc94770007dda54cF920
09BFF0dE90c06F603a09f","latest"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

---

eth_getBlockTransactionCountByHash

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

Parameters

1. `DATA`, 32 Bytes - hash of a block.

Example Parameters

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

`QUANTITY` - integer of the number of transactions in this block.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBlockTransactionCountByHash","params":["0xc9477000
7dda54cF92009BFF0dE90c06F603a09f"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xc" // 11
}
```

---

eth_getBlockTransactionCountByNumber

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

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.

Example Parameters

```
params: [
   '0xe8', // 232
]
```

Returns

`QUANTITY` - integer of the number of transactions in this block.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBlockTransactionCountByNumber","params":["0xe8"],"
id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xa" // 10
}
```

---

eth_getUncleCountByBlockHash

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

Parameters

1. `DATA`, 32 Bytes - hash of a block.

Example Parameters

```
params: [
   '0xc94770007dda54cF92009BFF0dE90c06F603a09f'
]
```

Returns

`QUANTITY` - integer of the number of uncles in this block.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleCountByBlockHash","params":["0xc94770007dda54
cF92009BFF0dE90c06F603a09f"],"id":1}'
```

```
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xc" // 1
}
```

---

eth_getUncleCountByBlockNumber

Returns the number of uncles in a block from a block matching the given block number.

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string "latest", "earliest" or "pending", see the default block parameter.

```
params: [
   '0xe8', // 232
]
```

Returns

`QUANTITY` - integer of the number of uncles in this block.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleCountByBlockNumber","params":["0xe8"],"id":1}
'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

---

eth_getCode

Returns code at a given address.

Parameters

1. `DATA`, 20 Bytes - address.
2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the [default block parameter](#).

Example Parameters

```
params: [
   '0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b',
   '0x2'  // 2
]
```

Returns

`DATA` - the code from the given address.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getCode","params":["0xa94f5374fce5edbc8e2a8697c153316
77e6ebf0b", "0x2"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060
078202905091905056"
}
```

---

eth_sign

The sign method calculates an Ethereum specific signature with: `sign(keccak256("\x19Ethereum Signed Message:\n" + len(message) + message)))`. By adding a prefix to the message makes the calculated signature recognisable as an Ethereum specific signature. This prevents misuse where a malicious DApp can sign arbitrary data (e.g. transaction) and use the signature to impersonate the victim.

Note the address to sign with must be unlocked.

Parameters

account, message

1. `DATA`, 20 Bytes - address.
2. `DATA`, N Bytes - message to sign.

Returns

`DATA`: Signature

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_sign","params":["0x9b2055d370f73ec7d8a03e965129118dc8
f5bf83", "0xdeadbeaf"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce34
81f10bea41f45ba9f709d39ce8325427b57afcfc994cee1b"
}
```

An example how to use solidity ecrecover to verify the signature calculated with `eth_sign` can be found here. The contract is deployed on the testnet Ropsten and Rinkeby.

---

eth_sendTransaction

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

Parameters

1. `Object` - The transaction object

- `from`: `DATA`, 20 Bytes - The address the transaction is send from.
- `to`: `DATA`, 20 Bytes - (optional when creating new contract) The address the transaction is directed to.
- `gas`: `QUANTITY` - (optional, default: 90000) Integer of the gas provided for the transaction execution. It will return unused gas.

- `gasPrice`: `QUANTITY` - (optional, default: To-Be-Determined) Integer of the gasPrice used for each paid gas
- `value`: `QUANTITY` - (optional) Integer of the value sent with this transaction
- `data`: `DATA` - The compiled code of a contract OR the hash of the invoked method signature and encoded parameters. For details see Ethereum Contract ABI
- `nonce`: `QUANTITY` - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

Example Parameters

```
params: [{
  "from": "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
  "to": "0xd46e8dd67c5d32be8058bb8eb970870f07244567",
  "gas": "0x76c0", // 30400
  "gasPrice": "0x9184e72a000", // 10000000000000
  "value": "0x9184e72a", // 2441406250
  "data":
"0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f072445675
"
}]
```

Returns

`DATA`, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available.
Use eth_getTransactionReceipt to get the contract address, after the transaction was mined, when you created a contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_sendTransaction","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

---

eth_sendRawTransaction

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

Parameters

1. `DATA`, The signed transaction data.

Example Parameters

```
params:
["0xd46e8dd67c5d32be8d46e8dd67c5d32be8058bb8eb970870f072445675058bb8eb970870f07244567
5"]
```

Returns

`DATA`, 32 Bytes - the transaction hash, or the zero hash if the transaction is not yet available.
Use [eth_getTransactionReceipt]() to get the contract address, after the transaction was mined, when you created a contract.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_sendRawTransaction","params":[{see above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331"
}
```

---

eth_call

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

Parameters

1. `Object` - The transaction call object

- `from`: `DATA`, 20 Bytes - (optional) The address the transaction is sent from.
- `to`: `DATA`, 20 Bytes - The address the transaction is directed to.

- `gas`: `QUANTITY` - (optional) Integer of the gas provided for the transaction execution. eth_call consumes zero gas, but this parameter may be needed by some executions.
- `gasPrice`: `QUANTITY` - (optional) Integer of the gasPrice used for each paid gas
- `value`: `QUANTITY` - (optional) Integer of the value sent with this transaction
- `data`: `DATA` - (optional) Hash of the method signature and encoded parameters. For details see Ethereum Contract ABI in the Solidity documentation

2. `QUANTITY|TAG` - integer block number, or the string `"latest"`, `"earliest"` or `"pending"`, see the default block parameter

Returns

`DATA` - the return value of executed contract.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_call","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x"
}
```

---

eth_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.

Parameters

See eth_call parameters, expect that all properties are optional. If no gas limit is specified geth uses the block gas limit from the pending block as an upper bound. As a result the returned estimate might not be enough to executed the call/transaction when the amount of gas is higher than the pending block gas limit.

Returns

`QUANTITY` - the amount of gas used.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_estimateGas","params":[{see
above}],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x5208" // 21000
}
```

---

eth_getBlockByHash

Returns information about a block by hash.

Parameters

1. `DATA`, 32 Bytes - Hash of a block.
2. `Boolean` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

Example Parameters

```
params: [
   '0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',
   true
]
```

Returns

`Object` - A block object, or `null` when no block was found:

- `number`: `QUANTITY` - the block number. `null` when its pending block.
- `hash`: `DATA`, 32 Bytes - hash of the block. `null` when its pending block.
- `parentHash`: `DATA`, 32 Bytes - hash of the parent block.
- `nonce`: `DATA`, 8 Bytes - hash of the generated proof-of-work. `null` when its pending block.

- `sha3Uncles`: `DATA`, 32 Bytes - SHA3 of the uncles data in the block.
- `logsBloom`: `DATA`, 256 Bytes - the bloom filter for the logs of the block. `null` when its pending block.
- `transactionsRoot`: `DATA`, 32 Bytes - the root of the transaction trie of the block.
- `stateRoot`: `DATA`, 32 Bytes - the root of the final state trie of the block.
- `receiptsRoot`: `DATA`, 32 Bytes - the root of the receipts trie of the block.
- `miner`: `DATA`, 20 Bytes - the address of the beneficiary to whom the mining rewards were given.
- `difficulty`: `QUANTITY` - integer of the difficulty for this block.
- `totalDifficulty`: `QUANTITY` - integer of the total difficulty of the chain until this block.
- `extraData`: `DATA` - the "extra data" field of this block.
- `size`: `QUANTITY` - integer the size of this block in bytes.
- `gasLimit`: `QUANTITY` - the maximum gas allowed in this block.
- `gasUsed`: `QUANTITY` - the total used gas by all transactions in this block.
- `timestamp`: `QUANTITY` - the unix timestamp for when the block was collated.
- `transactions`: `Array` - Array of transaction objects, or 32 Bytes transaction hashes depending on the last given parameter.
- `uncles`: `Array` - Array of uncle hashes.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBlockByHash","params":["0xe670ec64341771606e55d6b4
ca35a1a6b75ee3d5145a99d05921026d1527331", true],"id":1}'

// Result
{
"id":1,
"jsonrpc":"2.0",
"result": {
    "number": "0x1b4", // 436
    "hash": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",
    "parentHash":
"0x9646252be9520f6e71339a8df9c55e4d7619deeb018d2a3f2d21fc165dde5eb5",
    "nonce": "0xe04d296d2460cfb8472af2c5fd05b5a214109c25688d3704aed5484f9a7792f2",
    "sha3Uncles":
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "logsBloom": "0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331",
    "transactionsRoot":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "stateRoot":
"0xd5855eb08b3387c0af375e9cdb6acfc05eb8f519e419b874b6ff2ffda7ed1dff",
    "miner": "0x4e65fda2159562a496f9f3522f89122a3088497a",
    "difficulty": "0x027f07", // 163591
    "totalDifficulty":  "0x027f07", // 163591
    "extraData":
"0x0000000000000000000000000000000000000000000000000000000000000000",
    "size":  "0x027f07", // 163591
    "gasLimit": "0x9f759", // 653145
    "gasUsed": "0x9f759", // 653145
    "timestamp": "0x54e34e8e" // 1424182926
    "transactions": [{...},{ ... }]
    "uncles": ["0x1606e5...", "0xd5145a9..."]
  }
}
```

---

eth_getBlockByNumber

Returns information about a block by block number.

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `Boolean` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

Example Parameters

```
params: [
   '0x1b4', // 436
   true
]
```

Returns

See eth_getBlockByHash

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getBlockByNumber","params":["0x1b4", true],"id":1}'
```

Result see eth_getBlockByHash

---

eth_getTransactionByHash

Returns the information about a transaction requested by transaction hash.

Parameters

1. `DATA`, 32 Bytes - hash of a transaction

Example Parameters

```
params: [
   "0x88df016429689c079f3b2f6ad39fa052532c56795b733da78a91ebe6a713944b"
]
```

Returns

`Object` - A transaction object, or `null` when no transaction was found:

- `blockHash`: `DATA`, 32 Bytes - hash of the block where this transaction was in. `null` when its pending.
- `blockNumber`: `QUANTITY` - block number where this transaction was in. `null` when its pending.
- `from`: `DATA`, 20 Bytes - address of the sender.
- `gas`: `QUANTITY` - gas provided by the sender.
- `gasPrice`: `QUANTITY` - gas price provided by the sender in Wei.
- `hash`: `DATA`, 32 Bytes - hash of the transaction.
- `input`: `DATA` - the data send along with the transaction.
- `nonce`: `QUANTITY` - the number of transactions made by the sender prior to this one.
- `to`: `DATA`, 20 Bytes - address of the receiver. `null` when its a contract creation transaction.
- `transactionIndex`: `QUANTITY` - integer of the transaction's index position in the block. `null` when its pending.
- `value`: `QUANTITY` - value transferred in Wei.
- `v`: `QUANTITY` - ECDSA recovery id
- `r`: `QUANTITY` - ECDSA signature r
- `s`: `QUANTITY` - ECDSA signature s

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionByHash","params":["0x88df016429689c079f
3b2f6ad39fa052532c56795b733da78a91ebe6a713944b"],"id":1}'

// Result
{
  "jsonrpc":"2.0",
  "id":1,
  "result":{
    "blockHash":"0x1d59ff54b1eb26b013ce3cb5fc9dab3705b415a67127a003c3e61eb445bb8df2",
    "blockNumber":"0x5daf3b", // 6139707
    "from":"0xa7d9ddbe1f17865597fbd27ec712455208b6b76d",
    "gas":"0xc350", // 50000
    "gasPrice":"0x4a817c800", // 20000000000
    "hash":"0x88df016429689c079f3b2f6ad39fa052532c56795b733da78a91ebe6a713944b",
    "input":"0x68656c6c6f21",
    "nonce":"0x15", // 21
    "to":"0xf02c1c8e6114b1dbe8937a39260b5b0a374432bb",
    "transactionIndex":"0x41", // 65
    "value":"0xf3dbb76162000", // 4290000000000000
    "v":"0x25", // 37
    "r":"0x1b5e176d927f8e9ab405058b2d2457392da3e20f328b16ddabcebc33eaac5fea",
    "s":"0x4ba69724e8f69de52f0125ad8b3c5c2cef33019bac3249e2c0a2192766d1721c"
  }
}
```

---

eth_getTransactionByBlockHashAndIndex

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

Parameters

1. `DATA`, 32 Bytes - hash of a block.
2. `QUANTITY` - integer of the transaction index position.

Example Parameters

```
params: [
   '0xe670ec64341771606e55d6b4ca35a1a6b75ee3d5145a99d05921026d1527331',
   '0x0' // 0
]
```

Returns

See [eth_getTransactionByHash]

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionByBlockHashAndIndex","params":["0xc6ef2
fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b", "0x0"],"id":1}'
```

Result see eth_getTransactionByHash

---

eth_getTransactionByBlockNumberAndIndex

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

Parameters

1. `QUANTITY|TAG` - a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `QUANTITY` - the transaction index position.

Example Parameters

```
params: [
   '0x29c', // 668
   '0x0' // 0
]
```

Returns

See eth_getTransactionByHash

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionByBlockNumberAndIndex","params":["0x29c
", "0x0"],"id":1}'
```

Result see eth_getTransactionByHash

---

eth_getTransactionReceipt

Returns the receipt of a transaction by transaction hash.

Note That the receipt is not available for pending transactions.

Parameters

1. `DATA`, 32 Bytes - hash of a transaction

Example Parameters

```
params: [
   '0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238'
]
```

Returns

`Object` - A transaction receipt object, or `null` when no receipt was found:

- `transactionHash` : `DATA`, 32 Bytes - hash of the transaction.
- `transactionIndex`: `QUANTITY` - integer of the transaction's index position in the block.
- `blockHash`: `DATA`, 32 Bytes - hash of the block where this transaction was in.
- `blockNumber`: `QUANTITY` - block number where this transaction was in.
- `from`: `DATA`, 20 Bytes - address of the sender.
- `to`: `DATA`, 20 Bytes - address of the receiver. null when it's a contract creation transaction.
- `cumulativeGasUsed` : `QUANTITY` - The total amount of gas used when this transaction was executed in the block.
- `gasUsed` : `QUANTITY` - The amount of gas used by this specific transaction alone.
- `contractAddress` : `DATA`, 20 Bytes - The contract address created, if the transaction was a contract creation, otherwise `null`.
- `logs`: `Array` - Array of log objects, which this transaction generated.
- `logsBloom`: `DATA`, 256 Bytes - Bloom filter for light clients to quickly retrieve related logs.

It also returns *either* :

- `root` : `DATA` 32 bytes of post-transaction stateroot (pre Byzantium)
- `status`: `QUANTITY` either `1` (success) or `0` (failure)

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getTransactionReceipt","params":["0xb903239f8543d04b5
dc1ba6579132b143087c68db1b2168786408fcbce568238"],"id":1}'

// Result
{
"id":1,
"jsonrpc":"2.0",
"result": {
     transactionHash:
'0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238',
     transactionIndex:  '0x1', // 1
     blockNumber: '0xb', // 11
     blockHash: '0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b',
     cumulativeGasUsed: '0x33bc', // 13244
     gasUsed: '0x4dc', // 1244
     contractAddress: '0xb60e8dd61c5d32be8058bb8eb970870f07233155', // or null, if
none was created
     logs: [{
         // logs as returned by getFilterLogs, etc.
     }, ...],
     logsBloom: "0x00...0", // 256 byte bloom filter
     status: '0x1'
  }
}
```

eth_pendingTransactions

Returns the pending transactions list.

Parameters

none

Returns

`Array` - A list of pending transactions.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_pendingTransactions","params":[],"id":1}'

// Result
{
"id":1,
"jsonrpc":"2.0",
"result": [{
    blockHash: '0x0000000000000000000000000000000000000000000000000000000000000000',
    blockNumber: null,
    from: '0x28bdb9c230f4d5e45435e4d006326ee32e46cb31',
    gas: '0x204734',
    gasPrice: '0x4a817c800',
    hash: '0x8dfa6a59307a490d672494a171feee09db511f05e9c097e098edc2881f9ca4f6',
    input: '0x6080604052600',
    nonce: '0x12',
    to: null,
    transactionIndex: '0x0',
    value: '0x0',
    v: '0x3d',
    r: '0xaabc9ddafffb2ae0bac4107697547d22d9383667d9e97f5409dd6881ce08f13f',
    s: '0x69e43116be8f842dcd4a0b2f760043737a59534430b762317db21d9ac8c5034'
   },....,{
    blockHash: '0x0000000000000000000000000000000000000000000000000000000000000000',
    blockNumber: null,
    from: '0x28bdb9c230f4d5e45435e4d006326ee32e487b31',
    gas: '0x205940',
    gasPrice: '0x4a817c800',
    hash: '0x8e4340ea3983d86e4b6c44249362f716ec9e09849ef9b6e3321140581d2e4dac',
    input: '0xe4b6c4424936',
    nonce: '0x14',
    to: null,
    transactionIndex: '0x0',
    value: '0x0',
    v: '0x3d',
    r: '0x1ec191ef20b0e9628c4397665977cbe7a53a263c04f6f185132b77fa0fd5ca44',
    s: '0x8a58e00c63e05cfeae4f1cf19f05ce82079dc4d5857e2cc281b7797d58b5faf'
   }]
}
```

---

eth_getUncleByBlockHashAndIndex

Returns information about a uncle of a block by hash and uncle index position.

Parameters

1. `DATA`, 32 Bytes - hash a block.
2. `QUANTITY` - the uncle's index position.

```
params: [
   '0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b',
   '0x0' // 0
]
```

Returns

See eth_getBlockByHash

Example

```
// Request
```

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleByBlockHashAndIndex","params":["0xc6ef2fc5426
d6ad6fd9e2a26abeab0aa2411b7ab17f30a99d3cb96aed1d1055b", "0x0"],"id":1}'
```

Result see eth_getBlockByHash

Note: An uncle doesn't contain individual transactions.

---

eth_getUncleByBlockNumberAndIndex

Returns information about a uncle of a block by number and uncle index position.

Parameters

1. `QUANTITY|TAG` - a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `QUANTITY` - the uncle's index position.

Example Parameters

```
params: [
   '0x29c', // 668
   '0x0' // 0
]
```

Returns

See eth_getBlockByHash

Note: An uncle doesn't contain individual transactions.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getUncleByBlockNumberAndIndex","params":["0x29c",
"0x0"],"id":1}'
```

Result see eth_getBlockByHash

---

eth_getCompilers (DEPRECATED)

Returns a list of available compilers in the client.

Parameters

none

Returns

`Array` - Array of available compilers.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getCompilers","params":[],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": ["solidity", "lll", "serpent"]
}
```

---

eth_compileSolidity (DEPRECATED)

Returns compiled solidity code.

Parameters

1. `String` - The source code.

Example Parameters

```
params: [
   "contract test { function multiply(uint a) returns(uint d) {   return a * 7;   }
}",
]
```

Returns

`DATA` - The compiled source code.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_compileSolidity","params":["contract test { function
multiply(uint a) returns(uint d) {   return a * 7;   } }"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
      "code":
"0x605880600c6000396000f3006000357c01000000000000000000000000000000000000000000000000
0000000090048063c6888fa114602e57005b603d6004803590602001506047565b8060005260206000f35
b6000600782029050606053565b91905056",
      "info": {
        "source": "contract test {\n   function multiply(uint a) constant
returns(uint d) {\n       return a * 7;\n   }\n}\n",
        "language": "Solidity",
        "languageVersion": "0",
        "compilerVersion": "0.9.19",
        "abiDefinition": [
          {
            "constant": true,
            "inputs": [
              {
                "name": "a",
                "type": "uint256"
              }
            ],
            "name": "multiply",
            "outputs": [
              {
                "name": "d",
                "type": "uint256"
              }
            ],
            "type": "function"
          }
        ],
        "userDoc": {
          "methods": {}
        },
        "developerDoc": {
          "methods": {}
        }
      }

}
```

eth_compileLLL (DEPRECATED)

Returns compiled LLL code.

Parameters

1. `String` - The source code.

Example Parameters

```
params: [
   "(returnlll (suicide (caller)))",
]
```

Returns

`DATA` - The compiled source code.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_compileLLL","params":["(returnlll
(suicide (caller)))"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b60216004
35602b565b806000526020600f35b6000816007029050919050 56" // the compiled source code
}
```

---

eth_compileSerpent (DEPRECATED)

Returns compiled serpent code.

Parameters

1. `String` - The source code.

Example Parameters

```
params: [
   "/* some serpent */",
]
```

Returns

`DATA` - The compiled source code.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_compileSerpent","params":["/*
some serpent */"],"id":1}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0x603880600c6000396000f3006001600060e060020a600035048063c6888fa114601857005b60216004
35602b565b8060005260206000f35b6000816007029050919050 56" // the compiled source code
}
```

---

eth_newFilter

Creates a filter object, based on filter options, to notify when the state changes (logs). To check if the state has changed, call eth_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)"

Parameters

1. `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. Topics are order-dependent. Each topic can also be an array of DATA with "or" options.

Example Parameters

```
params: [{
  "fromBlock": "0x1",
  "toBlock": "0x2",
  "address": "0x8888f1f195afa192cfee860698584c030f4c9db1",
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b",
null, ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b",
"0x000000000000000000000000aff3454fce5edbc8cca8697c15331677e6ebccc"]]
}]
```

Returns

`QUANTITY` - A filter id.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_newFilter","params":[{"topics":["0x000000000000000000
00000000000000000000000000000000000000012341234"]}],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 1
}
```

---

eth_newBlockFilter

Creates a filter in the node, to notify when a new block arrives. To check if the state has changed, call eth_getFilterChanges.

Parameters

None

Returns

`QUANTITY` - A filter id.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_newBlockFilter","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":  "2.0",
  "result": "0x1" // 1
}
```

---

eth_newPendingTransactionFilter

Creates a filter in the node, to notify when new pending transactions arrive. To check if the state has changed, call eth_getFilterChanges.

Parameters

None

Returns

`QUANTITY` - A filter id.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_newPendingTransactionFilter","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":  "2.0",
  "result": "0x1" // 1
}
```

---

eth_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 eth_getFilterChanges for a period of time.

Parameters

1. `QUANTITY` - The filter id.

Example Parameters

```
params: [
  "0xb" // 11
]
```

Returns

`Boolean` - `true` if the filter was successfully uninstalled, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_uninstallFilter","params":["0xb"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

---

eth_getFilterChanges

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

Parameters

1. `QUANTITY` - the filter id.

Example Parameters

```
params: [
  "0x16" // 22
]
```

Returns

`Array` - Array of log objects, or an empty array if nothing has changed since last poll.

- For filters created with `eth_newBlockFilter` the return are block hashes (`DATA`, 32 Bytes), e.g. `["0x3454645634534..."]`.
- For filters created with `eth_newPendingTransactionFilter` the return are transaction hashes (`DATA`, 32 Bytes), e.g. `["0x6345343454645..."]`.
- For filters created with `eth_newFilter` logs are objects with following params:
    - `removed`: `TAG` - `true` when the log was removed, due to a chain reorganization. `false` if its a valid log.
    - `logIndex`: `QUANTITY` - integer of the log index position in the block. `null` when its pending log.
    - `transactionIndex`: `QUANTITY` - integer of the transactions index position log was created from. `null` when its pending log.
    - `transactionHash`: `DATA`, 32 Bytes - hash of the transactions this log was created from. `null` when its pending log.
    - `blockHash`: `DATA`, 32 Bytes - hash of the block where this log was in. `null` when its pending. `null` when its pending log.
    - `blockNumber`: `QUANTITY` - the block number where this log was in. `null` when its pending. `null` when its pending log.
    - `address`: `DATA`, 20 Bytes - address from which this log originated.
    - `data`: `DATA` - contains the non-indexed arguments of the log.
    - `topics`: `Array of DATA` - Array of 0 to 4 32 Bytes `DATA` of indexed log arguments. (In *solidity*: The first topic is the *hash* of the signature of the event (e.g. `Deposit(address,bytes32,uint256)`), except you declared the event with the `anonymous` specifier.)

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getFilterChanges","params":["0x16"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": [{
    "logIndex": "0x1", // 1
    "blockNumber":"0x1b4", // 436
    "blockHash": "0x8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcfdf829c5a142f1fccd7d",
    "transactionHash":
"0xdf829c5a142f1fccd7d8216c5785ac562ff41e2dcfdf5785ac562ff41e2dcf",
    "transactionIndex": "0x0", // 0
    "address": "0x16c5785ac562ff41e2dcfdf829c5a142f1fccd7d",
    "data":"0x0000000000000000000000000000000000000000000000000000000000000000",
    "topics": ["0x59ebeb90bc63057b6515673c3ecf9438e5058bca0f92585014eced636878c9a5"]
    },{
      ...
    }]
}
```

---

eth_getFilterLogs

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

Parameters

1. `QUANTITY` - The filter id.

Example Parameters

```
params: [
  "0x16" // 22
]
```

Returns

See eth_getFilterChanges

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getFilterLogs","params":["0x16"],"id":74}'
```

Result see eth_getFilterChanges

---

eth_getLogs

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

Parameters

1. `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. Topics are order-dependent. Each topic can also be an array of DATA with "or" options.
- `blockhash`: `DATA`, 32 Bytes - (optional) With the addition of EIP-234 (Geth >= v1.8.13 or Parity >= v2.1.0), `blockHash` is a new filter option which restricts the logs returned to the single block with the 32-byte hash `blockHash`. Using `blockHash` is equivalent to `fromBlock` = `toBlock` = the block number with hash `blockHash`. If `blockHash` is present in the filter criteria, then neither `fromBlock` nor `toBlock` are allowed.

Example Parameters

```
params: [{
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]
}]
```

Returns

See [eth_getFilterChanges]

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getLogs","params":[{"topics":["0x000000000000000000000
0000a94f5374fce5edbc8e2a8697c15331677e6ebf0b"]}],"id":74}'
```

Result see [eth_getFilterChanges]

---

eth_getWork

Returns the hash of the current block, the seedHash, and the boundary condition to be met ("target").

Parameters

none

Returns

`Array` - Array with the following properties:

1. `DATA`, 32 Bytes - current block header pow-hash
2. `DATA`, 32 Bytes - the seed hash used for the DAG.
3. `DATA`, 32 Bytes - the boundary condition ("target"), 2^256 / difficulty.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getWork","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": [
      "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
      "0x5EED00000000000000000000000000005EED0000000000000000000000000000",
      "0xd1ff1c01710000000000000000000000d1ff1c01710000000000000000000000"
    ]
}
```

---

eth_submitWork

Used for submitting a proof-of-work solution.

Parameters

1. `DATA`, 8 Bytes - The nonce found (64 bits)
2. `DATA`, 32 Bytes - The header's pow-hash (256 bits)

3. `DATA`, 32 Bytes - The mix digest (256 bits)

Example Parameters

```
params: [
  "0x0000000000000001",
  "0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
  "0xD1FE5700000000000000000000000000D1FE5700000000000000000000000000"
]
```

Returns

`Boolean` - returns `true` if the provided solution is valid, otherwise `false`.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"eth_submitWork",
"params":["0x0000000000000001",
"0x1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
"0xD1GE5700000000000000000000000000D1GE5700000000000000000000000000"],"id":73}'

// Result
{
  "id":73,
  "jsonrpc":"2.0",
  "result": true
}
```

---

eth_submitHashrate

Used for submitting mining hashrate.

Parameters

1. `Hashrate`, a hexadecimal string representation (32 bytes) of the hash rate
2. `ID`, String - A random hexadecimal(32 bytes) ID identifying the client

Example Parameters

```
params: [
  "0x0000000000000000000000000000000000000000000000000000000000500000",
  "0x59daa26581d0acd1fce254fb7e85952f4c09d0915afd33d3886cd914bc7d283c"
]
```

Returns

`Boolean` - returns `true` if submitting went through succesfully and `false` otherwise.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"eth_submitHashrate",
"params":["0x0000000000000000000000000000000000000000000000000000000000500000",
"0x59daa26581d0acd1fce254fb7e85952f4c09d0915afd33d3886cd914bc7d283c"],"id":73}'

// Result
{
  "id":73,
  "jsonrpc":"2.0",
  "result": true
}
```

---

eth_getProof

Returns the account- and storage-values of the specified account including the Merkle-proof.

getProof-Parameters

1. `DATA`, 20 bytes - address of the account or contract
2. `ARRAY`, 32 Bytes - array of storage-keys which should be proofed and included. See eth_getStorageAt
3. `QUANTITY|TAG` - integer block number, or the string "latest" or "earliest", see the default block parameter

Example Parameters

```
params:
["0x1234567890123456789012345678901234567890",["0x00000000000000000000000000000000000
00000000000000000000000000000","0x000000000000000000000000000000000000000000000000000
0000000000001"],"latest"]
```

getProof-Returns

Returns `Object` - A account object:
`balance`: `QUANTITY` - the balance of the account. See eth_getBalance

`codeHash`: `DATA`, 32 Bytes - hash of the code of the account. For a simple Account without code it will return "0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470"
`nonce`: `QUANTITY`, - nonce of the account. See eth_getTransactionCount
`storageHash`: `DATA`, 32 Bytes - SHA3 of the StorageRoot. All storage will deliver a MerkleProof starting with this rootHash.
`accountProof`: `ARRAY` - Array of rlp-serialized MerkleTree-Nodes, starting with the stateRoot-Node, following the path of the SHA3 (address) as key.
`storageProof`: `ARRAY` - Array of storage-entries as requested. Each entry is a object with these properties:
`key`: `QUANTITY` - the requested storage key `value`: `QUANTITY` - the storage value `proof`: `ARRAY` - Array of rlp-serialized MerkleTree-Nodes, starting with the storageHash-Node, following the path of the SHA3 (key) as path.

getProof-Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_getProof","params":["0x123456789012345678901234567890
1234567890",["0x0000000000000000000000000000000000000000000000000000000000000000","0x
0000000000000000000000000000000000000000000000000000000000000001"],"latest"],"id":1}'
-H "Content-type:application/json" http://localhost:8545

// Result
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "address": "0x1234567890123456789012345678901234567890",
    "accountProof": [

"0xf90211a090dcaf88c40c7bbc95a912cbdde67c175767b31173df9ee4b0d733bfdd511c43a0babe369f
6b12092f49181ae04ca173fb68d1a5456f18d20fa32cba73954052bda0473ecf8a7e36a829e75039a3b05
5e51b8332cbf03324ab4af2066bbd6fbf0021a0bbda34753d7aa6c38e603f360244e8f59611921d9e1f12
8372fec0d586d4f9e0a04e44caecff45c9891f74f6a2156735886eedf6f1a733628ebc802ec79d844648a
0a5f3f2f7542148c973977c8a1e154c4300fec92f755f7846f1b734d3ab1d90e7a0e823850f50bf72baae
9d1733a36a444ab65d0a6faaba404f0583ce0ca4dad92da0f7a00cbe7d4b30b11faea3ae61b7f1f2b315b
61d9f6bd68bfe587ad0eeceb721a07117ef9fc932f1a88e908eaead8565c19b5645dc9e5b1b6e841c5edb
dfd71681a069eb2de283f32c11f859d7bcf93da23990d3e662935ed4d6b39ce3673ec84472a0203d26456
312bbc4da5cd293b75b840fc5045e493d6f904d180823ec22bfed8ea09287b5c21f2254af4e64fca76acc
5cd87399c7f1ede818db4326c98ce2dc2208a06fc2d754e304c48ce6a517753c62b1a9c1d5925b8970748
6d7fc08919e0a94eca07b1c54f15e299bd58bdfef9741538c7828b5d7d11a489f9c20d052b3471df475a0
51f9dd3739a927c89e357580a4c97b40234aa01ed3d5e0390dc982a7975880a0a089d613f26159af43616
fd9455bb461f4869bfede26f2130835ed067a8b967bfb80",

"0xf90211a0395d87a95873cd98c21cf1df9421af03f7247880a2554e20738eec2c7507a494a0bcf65463
39a1e7e14eb8fb572a968d217d2a0d1f3bc4257b22ef5333e9e4433ca012ae12498af8b2752c99efce07f
3feef8ec910493be749acd63822c3558e6671a0dbf51303afdc36fc0c2d68a9bb05dab4f4917e7531e4a3
7ab0a153472d1b86e2a0ae90b50f067d9a2244e3d975233c0a0558c39ee152969f6678790abf773a9621a
01d65cd682cc1be7c5e38d8da5c942e0a73eeaef10f387340a40a106699d494c3a06163b53d956c555443
90c13634ea9aa75309f4fd866f312586942daf0f60fb37a058a52c1e858b1382a8893eb9c1f111f266eb9
e21e6137aff0dddea243a567000a037b4b100761e02de63ea5f1fcfcf43e81a372dafb4419d126342136d
329b7a7ba032472415864b08f808ba4374092003c8d7c40a9f7f9fe9cc8291f62538e1cc14a074e238ff5
ec96b810364515551344100138916594d6af966170ff326a092fab0a0d31ac4eef14a79845200a496662e
92186ca8b55e29ed0f9f59dbc6b521b116fea090607784fe738458b63c1942bba7c0321ae77e18df4961b
2bc66727ea996464ea078f757653c1b63f72aff3dcc3f2a2e4c8cb4a9d36d1117c742833c84e20de994a0
f78407de07f4b4cb4f899dfb95eedeb4049aeb5fc1635d65cf2f2f4dfd25d1d7a0862037513ba9d45354d
d3e36264aceb2b862ac79d2050f14c95657e43a51b85c80",

"0xf90171a04ad705ea7bf04339fa36b124fa221379bd5a38ffe9a6112cb2d94be3a437b879a08e45b5f7
2e8149c01efcb71429841d6a8879d4bbe27335604a5bff8dfdf85dcea00313d9b2f7c03733d6549ea3b81
0e5262ed844ea12f70993d87d3e0f04e3979ea0b59e3cdd6750fa8b15164612a5cb6567cdfb386d4e0137
fccee5f35ab55d0efda0fe6db56e42f2057a071c980a778d9a0b61038f269dd74a0e90155b3f40f14364a
08538587f2378a0849f9608942cf481da4120c360f8391bbcc225d811823c6432a026eac94e755534e16f
9552e73025d6d9c30d1d7682a4cb5bd7741ddabfd48c50a041557da9a74ca68da793e743e81e2029b2835
e1cc16e9e25bd0c1e89d4ccad6980a041dda0a40a21ade3a20fcd1a4abb2a42b74e9a32b02424ff8db4ea
708a5e0fb9a09aaf8326a51f613607a8685f57458329b41e938bb761131a5747e066b81a0a16808080a02
2e6cef138e16d2272ef58434ddf49260dc1de1f8ad6dfca3da5d2a92aaaadc58080",

"0xf851808080a009833150c367df138f1538689984b8a84fc55692d3d41fe4d1e5720ff5483a69808080
80808080808080a0a319c1c415b271afc0adcb664e67738d103ac168e0bc0b7bd2da7966165cb9518080"
    ],
    "balance": "0x0",
    "codeHash": "0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470",
    "nonce": "0x0",
    "storageHash":
"0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421",
    "storageProof": [
      {
        "key": "0x0000000000000000000000000000000000000000000000000000000000000000",
        "value": "0x0",
        "proof": []
      },
      {
        "key": "0x0000000000000000000000000000000000000000000000000000000000000001",
        "value": "0x0",
        "proof": []
      }
    ]
  }
}
```

---

db_putString

Stores a string in the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` - Database name.
2. `String` - Key name.
3. `String` - String to store.

Example Parameters

```
params: [
  "testDB",
  "myKey",
  "myString"
]
```

Returns

`Boolean` - returns `true` if the value was stored, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"db_putString","params":["testDB","myKey","myString"],"id"
:73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

---

db_getString

Returns string from the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` - Database name.
2. `String` - Key name.

Example Parameters

```
params: [
  "testDB",
  "myKey",
]
```

Returns

`String` - The previously stored string.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"db_getString","params":["testDB","myKey"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "myString"
}
```

---

db_putHex

Stores binary data in the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` - Database name.
2. `String` - Key name.
3. `DATA` - The data to store.

Example Parameters

```
params: [
  "testDB",
  "myKey",
  "0x68656c6c6f20776f726c64"
]
```

Returns

`Boolean` - returns `true` if the value was stored, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"db_putHex","params":["testDB","myKey","0x68656c6c6f20776f
726c64"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

---

db_getHex

Returns binary data from the local database.

Note this function is deprecated and will be removed in the future.

Parameters

1. `String` - Database name.
2. `String` - Key name.

Example Parameters

```
params: [
  "testDB",
  "myKey",
]
```

Returns

`DATA` - The previously stored data.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"db_getHex","params":["testDB","myKey"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "0x68656c6c6f20776f726c64"
}
```

---

shh_version

Returns the current whisper protocol version.

Parameters

none

Returns

`String` - The current whisper protocol version

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_version","params":[],"id":67}'

// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "2"
}
```

---

shh_post

Sends a whisper message.

Parameters

1. `Object` - The whisper post object:

- `from`: `DATA`, 60 Bytes - (optional) The identity of the sender.
- `to`: `DATA`, 60 Bytes - (optional) The identity of the receiver. When present whisper will encrypt the message so that only the receiver can decrypt it.
- `topics`: `Array of DATA` - Array of `DATA` topics, for the receiver to identify messages.
- `payload`: `DATA` - The payload of the message.

- `priority`: `QUANTITY` - The integer of the priority in a range from ... (?).
- `ttl`: `QUANTITY` - integer of the time to live in seconds.

Example Parameters

```
params: [{
  from:
"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284
d442460f2998cd41858798ddfd4d661997d3940272b717b1",
  to:
"0x3e245533f97284d442460f2998cd41858798ddf04f96a5e25610293e42a73908e93ccc8c4d4dc0edcf
a9fa872f50cb214e08ebf61a0d4d661997d3940272b717b1",
  topics: ["0x776869737065722d636861742d636c69656e74",
"0x4d5a695276454c39425154466b61693532"],
  payload: "0x7b2274797065223a226d6",
  priority: "0x64",
  ttl: "0x64",
}]
```

Returns

`Boolean` - returns `true` if the message was send, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_post","params":[{"from":"0xc931d93e97ab07fe42d923478b
a2465f2..","topics":
["0x68656c6c6f20776f726c64"],"payload":"0x68656c6c6f20776f726c64","ttl":0x64,"priorit
y":0x64}],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

---

shh_newIdentity

Creates new whisper identity in the client.

Parameters

none

Returns

`DATA`, 60 Bytes - the address of the new identiy.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_newIdentity","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0xc931d93e97ab07fe42d923478ba2465f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca9007d5b
68aa497e4619ac10aa3b27726e1863c1fd9b570d99bbaf"
}
```

---

shh_hasIdentity

Checks if the client hold the private keys for a given identity.

Parameters

1. `DATA`, 60 Bytes - The identity address to check.

Example Parameters

```
params: [
"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284
d442460f2998cd41858798ddfd4d661997d3940272b717b1"
]
```

Returns

`Boolean` - returns `true` if the client holds the privatekey for that identity, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_hasIdentity","params":["0x04f96a5e25610293e42a73908e9
3ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d442460f2998cd41858798ddfd4d66
1997d3940272b717b1"],"id":73}'
```

```
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

---

shh_newGroup

Creates a new group.

Parameters

none

Returns

`DATA`, 60 Bytes - the address of the new group.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newGroup","params":[],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result":
"0xc65f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca90931d93e97ab07fe42d923478ba2407d5b
68aa497e4619ac10aa3b27726e1863c1fd9b570d99bbaf"
}
```

---

shh_addToGroup

Adds a whisper identity to the group.

Parameters

1. `DATA`, 60 Bytes - The identity address to add to a group.

Example Parameters

```
params: [
"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284
d442460f2998cd41858798ddfd4d661997d3940272b717b1"
]
```

Returns

`Boolean` - returns `true` if the identity was successfully added to the group, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_addToGroup","params":["0x04f96a5e25610293e42a73908e93
ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d442460f2998cd41858798ddfd4d661
997d3940272b717b1"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

---

shh_newFilter

Creates filter to notify, when client receives whisper message matching the filter options.

Parameters

1. `Object` - The filter options:

- `to`: `DATA`, 60 Bytes - (optional) Identity of the receiver. *When present it will try to decrypt any incoming message if the client holds the private key to this identity.*
- `topics`: `Array of DATA` - Array of `DATA` topics which the incoming message's topics should match. You can use the following combinations:
  - `[A, B] = A && B`
  - `[A, [B, C]] = A && (B || C)`
  - `[null, A, B] = ANYTHING && A && B` `null` works as a wildcard

Example Parameters

```
params: [{
   "topics": ['0x12341234bf4b564f'],
   "to":
"0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284
d442460f2998cd41858798ddfd4d661997d3940272b717b1"
}]
```

Returns

`QUANTITY` - The newly created filter.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newFilter","params":[{"topics":
['0x12341234bf4b564f'],"to": "0x2341234bf4b2341234bf4b564f..."}],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "0x7" // 7
}
```

---

shh_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 shh_getFilterChanges for a period of time.

Parameters

1. `QUANTITY` - The filter id.

Example Parameters

```
params: [
  "0x7" // 7
]
```

Returns

`Boolean` - `true` if the filter was successfully uninstalled, otherwise `false`.

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_uninstallFilter","params":["0x7"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

---

shh_getFilterChanges

Polling method for whisper filters. Returns new messages since the last call of this method.

Note calling the shh_getMessages method, will reset the buffer for this method, so that you won't receive duplicate messages.

Parameters

1. `QUANTITY` - The filter id.

Example Parameters

```
params: [
  "0x7" // 7
]
```

Returns

`Array` - Array of messages received since last poll:

- `hash`: `DATA`, 32 Bytes (?) - The hash of the message.
- `from`: `DATA`, 60 Bytes - The sender of the message, if a sender was specified.
- `to`: `DATA`, 60 Bytes - The receiver of the message, if a receiver was specified.
- `expiry`: `QUANTITY` - Integer of the time in seconds when this message should expire (?).
- `ttl`: `QUANTITY` - Integer of the time the message should float in the system in seconds (?).
- `sent`: `QUANTITY` - Integer of the unix timestamp when the message was sent.
- `topics`: `Array of DATA` - Array of `DATA` topics the message contained.
- `payload`: `DATA` - The payload of the message.

- `workProved`: `QUANTITY` - Integer of the work this message required before it was send (?).

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_getFilterChanges","params":["0x7"],"id":73}'

// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": [{
    "hash": "0x33eb2da77bf3527e28f8bf493650b1879b08c4f2a362beae4ba2f71bafcd91f9",
    "from": "0x3ec052fc33..",
    "to": "0x87gdf76g8d7fgdfg...",
    "expiry": "0x54caa50a", // 1422566666
    "sent": "0x54ca9ea2", // 1422565026
    "ttl": "0x64", // 100
    "topics": ["0x6578616d"],
    "payload": "0x7b2274797065223a226d657373616765222c2263686...",
    "workProved": "0x0"
    }]
}
```

---

shh_getMessages

Get all messages matching a filter. Unlike `shh_getFilterChanges` this returns all messages.

Parameters

1. `QUANTITY` - The filter id.

Example Parameters

```
params: [
  "0x7" // 7
]
```

Returns

See shh_getFilterChanges

Example

```
// Request
curl -X POST --data
'{"jsonrpc":"2.0","method":"shh_getMessages","params":["0x7"],"id":73}'
```

Result see shh_getFilterChanges