JSON-RPC API

Última edição: @corwintines ↗ , 17 de julho de 2023

Ver colaboradores

Nesta página

In order for a software application to interact with the Ethereum blockchain - either by reading blockchain data or sending transactions to the network - it must connect to an Ethereum node.

For this purpose, every Ethereum client implements a JSON-RPC specification ↗ , so there is a uniform set of methods that applications can rely on regardless of the specific node or client implementation.

JSON-RPC ↗ is a stateless, light-weight remote procedure call (RPC) protocol. It 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.

CLIENT IMPLEMENTATIONS

Ethereum clients each may utilize different programming languages when implementing the JSON-RPC specification. See individual client documentation for further details related to specific programming languages. We recommend checking the documentation of each client for the latest API support information.

CONVENIENCE LIBRARIES

While you may choose to interact directly with Ethereum clients via the JSON-RPC API, there are often easier options for dapp developers. Many JavaScript and backend API libraries exist to provide wrappers on top of the JSON-RPC API. With these libraries, developers can write intuitive, one-line methods in the programming language of their choice to initialize JSON-RPC requests (under the hood) that interact with Ethereum.

CONSENSUS CLIENT APIS

This page deals mainly with the JSON-RPC API used by Ethereum execution clients. However, consensus clients also have an RPC API that allows users to query information about the node, request Beacon blocks, Beacon state, and other consensus-related information directly from a node. This API is documented on the Beacon API webpage ↗ .

An internal API is also used for inter-client communication within a node - that is, it enables the consensus client and execution client to swap data. This is called the 'Engine API' and the specs are available on GitHub ↗ .

EXECUTION CLIENT SPEC

Read the full JSON-RPC API spec on GitHub ↗ .

CONVENTIONS

Hex value encoding

Two key data types get passed over JSON: unformatted byte arrays and quantities. Both are passed with a hex encoding but with different requirements for formatting.

Quantities

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

Here are some examples:

- 0×41 (65 in decimal)
- 0×400 (1024 in decimal)
- WRONG: 0x (should always have at least one digit - zero is "0×0")
- WRONG: 0×0400 (no leading zeroes allowed)
- WRONG: ff (must be prefixed 0x)

Unformatted data

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

Here are some examples:

- 0×41 (size 1, "A")
- 0×004200 (size 3, "\0B\0")
- 0x (size 0, "")
- WRONG: 0xf0f0f (must be even number of digits)
- WRONG: 004200 (must be prefixed 0x)

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 "safe"` - for the latest safe head block
- `String "finalized"` - for the latest finalized block
- `String "pending"` - for the pending state/transactions

EXAMPLES

On this page we provide examples of how to use individual JSON_RPC API endpoints using the command line tool, curl ↗ . These individual endpoint examples are found below in the Curl examples section. Further down the page, we also provide an end-to-end example for compiling and deploying a smart contract using a Geth node, the JSON_RPC API and curl.

CURL EXAMPLES

Examples of using the JSON_RPC API by making curl ↗ requests to an Ethereum node are provided below. Each example includes a description of the specific endpoint, its parameters, return type, and a worked example of how it should be used.

The curl requests might return an error message relating to the content type. This is because the `--data` option sets the content type to `application/x-www-form-urlencoded`. If your node does complain about this, 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.g. `127.0.0.1:8545`). A complete curl request including these additional data takes the following form:

```
curl -H "Content-Type: application/json" -X POST --data '{"jsonrpc":"2.0","method":"web3_clientVersion","params":[],"id":67}' 127.0.0
```

GOSSIP, STATE, HISTORY

A handful of core JSON-RPC methods require data from the Ethereum network, and fall neatly into three main categories: *Gossip, State, and History*. Use the links in these sections to jump to each method, or use the table of contents to explore the whole list of methods.

Gossip Methods

These methods track the head of the chain. This is how transactions make their way around the network, find their way into blocks, and how clients find out about new blocks.

- eth_blockNumber
- eth_sendRawTransaction

State Methods

Methods that report the current state of all the data stored. The "state" is like one big shared piece of RAM, and includes account balances, contract data, and gas estimations.

- eth_getBalance
- eth_getStorageAt

- eth_getTransactionCount
- eth_getCode
- eth_call
- eth_estimateGas

History Methods

Fetches historical records of every block back to genesis. This is like one large append-only file, and includes all block headers, block bodies, uncle blocks, and transaction receipts.

- eth_getBlockTransactionCountByHash
- eth_getBlockTransactionCountByNumber
- eth_getUncleCountByBlockHash
- eth_getUncleCountByBlockNumber
- eth_getBlockByHash
- eth_getBlockByNumber
- eth_getTransactionByHash
- eth_getTransactionByBlockHashAndIndex
- eth_getTransactionByBlockNumberAndIndex
- eth_getTransactionReceipt
- eth_getUncleByBlockHashAndIndex
- eth_getUncleByBlockNumberAndIndex

JSON-RPC API METHODS

web3_clientVersion

Returns the current client version.

Parameters

None

Returns

`String` - The current client version

Example

Copiar

```
// 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

```
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.

The full list of current network IDs is available at chainlist.org ↗ . Some common ones are:

- `1`: Ethereum Mainnet
- `5`: Goerli testnet
- `11155111`: Sepolia 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.

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. Note that this method is [not available in Geth ↗](#) .

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": "54"
}
```

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',
```

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": "0x407d73d8a49eeb85d32cf465507dd71d507100c1"
}
```

eth_chainId

Returns the chain ID used for signing replay-protected transactions.

Parameters

None

Returns

`chainId`, hexadecimal value as a string representing the integer of the current chain id.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_chainId","params":[],"id":67}'
// Result
{
  "id":67,
  "jsonrpc": "2.0",
  "result": "0x1"
}
```

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}'
//
{
  "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": "0x1dfd14000" // 8049999872 Wei
}
```

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": ["0x407d73d8a49eeb85d32cf465507dd71d507100c1"]
}
```

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":83}'
// Result
{
  "id":83,
  "jsonrpc": "2.0",
  "result": "0x4b7" // 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

```
params: ["0x407d73d8a49eeb85d32cf465507dd71d507100c1", "latest"]
```

Returns

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

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBalance","params":["0x407d73d8a49eeb85d32cf465507dd71d507100c1", "latest"],"id
// 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",
{"jsonrpc":"2.0","id":1,"result":"0x00000000000000000000000000000000000000000000000000000000000004d2"}
```

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["0×391694e7e0b0cce554cb130d723a9d27458f9298"] 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" + "0000000000000000000000000000000000000000000000000000000000000000
undefined
> web3.sha3(key, {"encoding": "hex"})
"0x6661e9d6d8b923d5bbaab1b96e1dd51ff6ea2a93520fdc9eb75d059238b8c5e9"
```

Now to fetch the storage:

```
curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getStorageAt", "params": ["0x295a70b2de5e3953354a6a8344e616ed314d7251", "0x6661
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000000000000162e"}
```

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

```
params: [
  "0x407d73d8a49eeb85d32cf465507dd71d507100c1",
  "latest", // state at the latest block
]
```

Returns

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

Example


```
    "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

```
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":["0xb903239f8543d04b5dc1ba6579132b143087
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xb" // 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.

```
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

```
params: ["0xb903239f8543d04b5dc1ba6579132b143087c68db1b2168786408fcbce568238"]
```

Returns

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

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getUncleCountByBlockHash","params":["0xb903239f8543d04b5dc1ba6579132b143087c68db1
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x1" // 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

```
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":["0xa94f5374fce5edbc8e2a8697c15331677e6ebf0b", "0x2"],"id":1}'
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0x600160008035811a818181146012578301005b601b6001356025565b8060005260206000f25b600060078202905091905056"
}
```

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 recognizable 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

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":["0x9b2055d370f73ec7d8a03e965129118dc8f5bf83", "0xdeadbeaf"],"id":
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9f709d39ce8325427b57a
}
```

eth_signTransaction

Signs a transaction that can be submitted to the network at a later time using with eth_sendRawTransaction.

Parameters

1. `Object` - The transaction object

- `from`: `DATA`, 20 Bytes - The address the transaction is sent 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, in Wei.
- `value`: `QUANTITY` - (optional) Integer of the value sent with this transaction, in Wei.
- `data`: `DATA` - The compiled code of a contract OR the hash of the invoked method signature and encoded parameters.
- `nonce`: `QUANTITY` - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

Returns

`DATA`, The signed transaction object.

Example

```
// Request
curl -X POST --data '{"id": 1,"jsonrpc": "2.0","method": "eth_signTransaction","params": [{"data":"0xd46e8dd67c5d32be8d46e8dd67c5d32b
// Result
{
    "id": 1,
    "jsonrpc": "2.0",
    "result": "0xa3f20717a250c2b0b729b7e5becbff67fdaef7e0699da4de7ca5895b02a170a12d887fd3b17bfdce3481f10bea41f45ba9f709d39ce8325427b5
}
```

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

Mostrar todos Copiar

```
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

Copiar

```
// 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.

Copiar

```
params: [
  "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_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, except 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

Copiar

```
// 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.

Copiar

```
params: [
  "0xdc0818cf78f21a8e70579cb46a43643f78291264dda342ae31049421c82d21ae",
  false,
]
```

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

Mostrar todos Copiar

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBlockByHash","params":["0xdc0818cf78f21a8e70579cb46a43643f78291264dda342ae310
// Result
{
{
"jsonrpc": "2.0",
"id": 1,
"result": {
    "difficulty": "0x4ea3f27bc",
```

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.

Copiar

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

```

Returns See eth_getBlockByHash

Example

Copiar

```
// 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

```
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 transactions 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":["0x88df016429689c079f3b2f6ad39fa052532c56795b733
// Result
{
  "jsonrpc":"2.0",
  "id":1,
  "result":{
    "blockHash":"0x1d59ff54b1eb26b013ce3cb5fc9dab3705b415a67127a003c3e61eb445bb8df2",
    "blockNumber":"0x5daf3b", // 6139707
```

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.

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

Returns See eth_getTransactionByHash

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getTransactionByBlockHashAndIndex","params":["0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2
```

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.

```
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

```
params: ["0x85d995eba9763907fdf35cd2034144dd9d53ce32cbec21349d4b12823c6860c5"]
```

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 transactions 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 its a contract creation transaction.
- `cumulativeGasUsed` : `QUANTITY` - The total amount of gas used when this transaction was executed in the block.
- `effectiveGasPrice` : `QUANTITY` - The sum of the base fee and tip paid per unit of gas.
- `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.
- `type`: `QUANTITY` - integer of the transaction type, `0x0` for legacy transactions, `0x1` for access list types, `0x2` for dynamic fees.

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":["0x85d995eba9763907fdf35cd2034144dd9d53ce32cbec
// Result
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "blockHash":
      "0xa957d47df264a31badc3ae823e10ac1d444b098d9b73d204c40426e57f47e8c3",
```

eth_getUncleByBlockHashAndIndex

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

Parameters

1. `DATA`, 32 Bytes - The hash of 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":["0xc6ef2fc5426d6ad6fd9e2a26abeab0aa2411b7a
```

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.

```
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

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

Returns compiled solidity code.

Parameters

1. `String` - The source code.

```
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(ui
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": {
      "code": "0x605880600c6000396000f3006000357c010000000000000000000000000000000000000000000000000000000090048063c6888fa114602e570
      "info": {
```

eth_compileLLL

Returns compiled LLL code.

Parameters

1. `String` - The source code.

```
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": "0x603880600c6000396000f300600160006e060020a600035048063c6888fa114601857005b6021600435602b565b8060005260206000f35b600081
}
```

eth_compileSerpent

Returns compiled serpent code.

Parameters

1. `String` - The source code.

```
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": "0x603880600c6000396000f300600160006e060020a600035048063c6888fa114601857005b6021600435602b565b8060005260206000f35b600081
}
```

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.

Mostrar todos Copiar

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

Returns `QUANTITY` - A filter id.

Example

Copiar

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_newFilter","params":[{"topics":["0x12341234"]}],"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

Copiar

```
// 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. Additionally Filters timeout when they aren't requested with eth_getFilterChanges for a period of time.

Parameters

1. `QUANTITY` - The filter id.

```
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.

```
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 one or more 32 Bytes 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**

Mostrar todos | Copiar

```
// 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
```

eth_getFilterLogs

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

Parameters

1. `QUANTITY` - The filter id.

Copiar

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

Returns See eth_getFilterChanges

Example

Copiar

```
// 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, **future**) With the addition of EIP-234, `blockHash` will be 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.

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

Returns See eth_getFilterChanges

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getLogs","params":[{"topics":["0x000000000000000000000000a94f5374fce5edbc8e2a8697
```

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",
```

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)

Copiar

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

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

Example

Copiar

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"eth_submitWork", "params":["0x0000000000000001", "0x1234567890abcdef1234567890abcdef
// 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 hashrate
2. `ID`, String - A random hexadecimal(32 bytes) ID identifying the client

Copiar

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

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

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0", "method":"eth_submitHashrate", "params":["0x0000000000000000000000000000000000000000000000000000000000000000",
// Result
{
  "id":73,
  "jsonrpc":"2.0",
  "result": true
}
```

db_putString (deprecated)

Stores a string in the local database.

Note this function is deprecated.

Parameters

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

```
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 (deprecated)

Returns string from the local database. **Note** this function is deprecated.

Parameters

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

```
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 (deprecated)

Stores binary data in the local database. **Note** this function is deprecated.

Parameters

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

```
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","0x68656c6c6f20776f726c64"],"id":73}'
// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

db_getHex (deprecated)

Returns binary data from the local database. **Note** this function is deprecated.

Parameters

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

```
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 (deprecated)

Returns the current whisper protocol version.

Note this function is deprecated.

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 (deprecated)

Sends a whisper message.

Note this function is deprecated.

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 rang from ... (?).
- `ttl`: `QUANTITY` - integer of the time to live in seconds.

```
  params: [
    {
      from: "0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d442460f2998cd41858798ddfd4d661997d39
      to: "0x3e245533f97284d442460f2998cd41858798ddf04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a0d4d661997d3940
      topics: [
        "0x776869737065722d636861742d636c69656e74",
        "0x4d5a695276454c39425154466b61693532",
      ],
      payload: "0x7b2274797065223a226d6",
```

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":"0xc931d93e97ab07fe42d923478ba2465f2..","topics": ["0x686
// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": true
}
```

shh_newIdentity (deprecated)

Creates new whisper identity in the client.

Note this function is deprecated.

Parameters None

Returns `DATA`, 60 Bytes - the address of the new identity.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newIdentity","params":[],"id":73}'
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": "0xc931d93e97ab07fe42d923478ba2465f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca9007d5b68aa497e4619ac10aa3b27726e1863c1fd9b
}
```

shh_hasIdentity (deprecated)

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

Note this function is deprecated.

Parameters

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

```
params: [
  "0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d442460f2998cd41858798ddfd4d661997d3940272b717
]
```

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":["0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

shh_newGroup (deprecated)

Note this function is deprecated.

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": "0xc65f283f440fd6cabea4dd7a2c807108f651b7135d1d6ca90931d93e97ab07fe42d923478ba2407d5b68aa497e4619ac10aa3b27726e1863c1fd9b
}
```

shh_addToGroup (deprecated)

Note this function is deprecated.

Parameters

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

```
params: [
  "0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d442460f2998cd41858798ddfd4d661997d3940272b717
]
```

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":["0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb2
// Result
{
  "id":1,
  "jsonrpc": "2.0",
  "result": true
}
```

shh_newFilter (deprecated)

Creates filter to notify, when client receives whisper message matching the filter options. **Note** this function is deprecated.

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
  -

```
params: [
  {
    topics: ["0x12341234bf4b564f"],
    to: "0x04f96a5e25610293e42a73908e93ccc8c4d4dc0edcfa9fa872f50cb214e08ebf61a03e245533f97284d442460f2998cd41858798ddfd4d661997d39402
  },
]
```

Returns `QUANTITY` - The newly created filter.

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"shh_newFilter","params":[{"topics": ['0x12341234bf4b564f'],"to": "0x2341234bf4b234123
// Result
{
  "id":1,
  "jsonrpc":"2.0",
  "result": "0x7" // 7
}
```

shh_uninstallFilter (deprecated)

Uninstalls a filter with given id. Should always be called when watch is no longer needed. Additionally Filters timeout when they aren't requested with shh_getFilterChanges for a period of time. **Note** this function is deprecated.

Parameters

1. `QUANTITY` - The filter id.

```
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 (deprecated)

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. **Note** this function is deprecated.

Parameters

1. `QUANTITY` - The filter id.

```
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..",
```

shh_getMessages (deprecated)

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

Note this function is deprecated.

Parameters

1. `QUANTITY` - The filter id.

```
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

USAGE EXAMPLE

Deploying a contract using JSON_RPC

This section includes a demonstration of how to deploy a contract using only the RPC interface. There are alternative routes to deploying contracts where this complexity is abstracted away—for example, using libraries built on top of the RPC interface such as web3.js ↗ and web3.py ↗ . These abstractions are generally easier to understand and less error-prone, but it is still helpful to understand what is happening under the hood.

The following is a straightforward smart contract called `Multiply7` that will be deployed using the JSON-RPC interface to an Ethereum node. This tutorial assumes the reader is already running a Geth node. More information on nodes and clients is available here. Please refer to individual client documentation to see how to start the HTTP JSON-RPC for non-Geth clients. Most clients default to serving on `localhost:8545`.

```
contract Multiply7 {
    event Print(uint);
    function multiply(uint input) returns (uint) {
```

```
            Print(input * 7);
            return input * 7;
        }
    }

```

The first thing to do is make sure the HTTP RPC interface is enabled. This means we supply Geth with the `--http` flag on startup. In

```
geth --http --dev console 2>>geth.log
```

This will start the HTTP RPC interface on `http://localhost:8545`.

We can verify that the interface is running by retrieving the Coinbase address and balance using curl ↗ . Please note that data in these examples will differ on your local node. If you want to try these commands, replace the request params in the second curl request with the result returned from the first.

```
curl --data '{"jsonrpc":"2.0","method":"eth_coinbase", "id":1}' -H "Content-Type: application/json" localhost:8545
{"id":1,"jsonrpc":"2.0","result":["0x9b1d35635cc34752ca54713bb99d38614f63c955"]}

curl --data '{"jsonrpc":"2.0","method":"eth_getBalance", "params": ["0x9b1d35635cc34752ca54713bb99d38614f63c955", "latest"], "id":2}' -H "(
{"id":2,"jsonrpc":"2.0","result":"0x1639e49bba16280000"}
```

Because numbers are hex encoded, the balance is returned in wei as a hex string. If we want to have the balance in ether as a number we can use web3 from the Geth console.

```
    web3.fromWei("0x1639e49bba16280000", "ether")
    // "410"

```

Now that there is some ether on our private development chain, we can deploy the contract. The first step is to compile the Multiply7 contract to byte code that can be sent to the EVM. To install solc, the Solidity compiler, follow the Solidity documentation ↗ . (You might want to use an older `solc` release to match the version of compiler used for our example ↗ .)

The next step is to compile the Multiply7 contract to byte code that can be send to the EVM.

```
echo 'pragma solidity ^0.4.16; contract Multiply7 { event Print(uint); function multiply(uint input) public returns (uint) { Print(input *

======= <stdin>:Multiply7 =======
Binary:
606060405234156100f57600080fd5b60eb8061001d6000396000f300606060405260043610603f576000357c0100000000000000000000000000000000000000000000000000000
```

Now that we have the compiled code we need to determine how much gas it costs to deploy it. The RPC interface has an `eth_estimateGas` method that will give us an estimate.

```
curl --data '{"jsonrpc":"2.0","method": "eth_estimateGas", "params": [{"from": "0x9b1d35635cc34752ca54713bb99d38614f63c955", "data": "0x606
{"jsonrpc":"2.0","id":5,"result":"0x1c31e"}
```

And finally deploy the contract.

```
curl --data '{"jsonrpc":"2.0","method": "eth_sendTransaction", "params": [{"from": "0x9b1d35635cc34752ca54713bb99d38614f63c955", "gas": "0x
{"id":6,"jsonrpc":"2.0","result":"0xe1f3095770633ab2b18081658bad475439f6a08c902d0915903bafff06e6febf"}
```

The transaction is accepted by the node and a transaction hash is returned. This hash can be used to track the transaction. The next step is to determine the address where our contract is deployed. Each executed transaction will create a receipt. This receipt contains various information about the transaction such as in which block the transaction was included and how much gas was used by the EVM. If a transaction creates a contract it will also contain the contract address. We can retrieve the receipt with the `eth_getTransactionReceipt` RPC method.

```
curl --data '{"jsonrpc":"2.0","method": "eth_getTransactionReceipt", "params": ["0xe1f3095770633ab2b18081658bad475439f6a08c902d0915903baff
{"jsonrpc":"2.0","id":7,"result":{"blockHash":"0x77b1a4f6872b9066312de3744f60020cbd8102af68b1f6512a05b7619d527a4f","blockNumber":"0x1","cor
```

Our contract was created on `0x4d03d617d700cf81935d7f797f4e2ae719648262`. A null result instead of a receipt means the transaction has not been included in a block yet. Wait for a moment and check if your miner is running and retry it.

Interacting with smart contracts

In this example we will be sending a transaction using `eth_sendTransaction` to the `multiply` method of the contract.

`eth_sendTransaction` requires several arguments, specifically `from`, `to` and `data`. `From` is the public address of our account, and `to` is the contract address. The `data` argument contains a payload that defines which method must be called and with which arguments. This is where the ABI (application binary interface) ↗ comes into play. The ABI is a JSON file that defines how to define and encode data for the EVM.

The bytes of the payload defines which method in the contract is called. This is the first 4 bytes from the Keccak hash over the function name and its argument types, hex encoded. The multiply function accepts an uint which is an alias for uint256. This leaves us with:

```
web3.sha3("multiply(uint256)").substring(0, 10)
// "0xc6888fa1"

```

The next step is to encode the arguments. There is only one uint256, say, the value 6. The ABI has a section which specifies how to encode uint256 types.

`int<M>: enc(X)` is the big-endian two's complement encoding of X, padded on the higher-order (left) side with 0xff for negative X and with zero > bytes for positive X such that the length is a multiple of 32 bytes.

This encodes to `0000000000000000000000000000000000000000000000000000000000000006`.

Combining the function selector and the encoded argument our data will be `0xc6888fa10000000000000000000000000000000000000000000000000000000000000006`.

This can now be sent to the node:

```
curl --data '{"jsonrpc":"2.0","method": "eth_sendTransaction", "params": [{"from": "0xeb85a5557e5bdc18ee1934a89d8bb402398ee26a", "to": "0x
{"id":8,"jsonrpc":"2.0","result":"0x759cf065cbc22e9d779748dc53763854e5376eea07409e590c990eafc0869d74"}
```

Since a transaction was sent, a transaction hash was returned. Retrieving the receipt gives:

Mostrar todos

```
{
   blockHash: "0xbf0a347307b8c63dd8c1d3d7cbdc0b463e6e7c9bf0a35be40393588242f01d55",
   blockNumber: 268,
   contractAddress: null,
   cumulativeGasUsed: 22631,
   gasUsed: 22631,
   logs: [{
```

```
8           address: "0x6ff93b4b46b41c0c3c9baee01c255d3b4675963d",
9           blockHash: "0xbf0a247307b8c63dd8c1d3d7cbdc0b463e6e7c9bf0a25be40393588242f01d55"
```

The receipt contains a log. This log was generated by the EVM on transaction execution and included in the receipt. The `multiply` function shows that the `Print` event was raised with the input times 7. Since the argument for the `Print` event was a uint256 we can decode it according to the ABI rules which will leave us with the expected decimal 42. Apart from the data it is worth noting that topics can be used to determine which event created the log:

```
1   web3.sha3("Print(uint256)")
2   // "24abdb5865df5079dcc5ac590ff6f01d5c16edbc5fab4e195d9febd1114503da"
3
```

This was just a brief introduction into some of the most common tasks, demonstrating direct usage of the JSON-RPC.

RELATED TOPICS

- JSON-RPC specification ↗
- Nodes and clients
- JavaScript APIs
- Backend APIs
- Execution clients

Voltar para o início ↑

PRÓXIMO
Data and analytics

ANTERIOR
Backend APIs

Stablecoins

Bloquear ETH

O que é Web3?

Contratos inteligentes

Consumo energético Ethereum

Roteiro Ethereum

Propostas de Melhoramento do Ethereum

A história do Ethereum

Documento técnico do Ethereum

Glossário Ethereum

Governação Ethereum

Pontes de Blockchain

Provas de conhecimento-zero

Centro de testes

Programadores

Começar

Documentação

Tutoriais

Aprenda usando programação

Configurar ambiente local

Ecossistema

Centro de Comunidade

Fundação Ethereum

Blog da Fundação Ethereum ↗

Programa de Suporte do Ecossistema ↗

Programa de recompensas de bug de Ethereum

Programas de concessão do ecossistema

Ativos da Marca Ethereum

Devcon ↗

Empresarial

Mainnet Ethereum

Ethereum privado

Empresarial

Acerca do ethereum.org

Quem somos

Empregos

Contribuir

Suporte para idiomas

Política de privacidade

Termos de Utilização

Política de Cookies

Contacto ↗