oasis.js Documentation

Oasis Labs Inc. <info@oasislabs.com>

Oct 30, 2020

Quickstart

oasis.js provides a collection of tools to interact with Secure Services running on the Oasis blockchain. If you're new to Oasis, first checkout the oasis docs. For developing services, check out the oasis-rs docs.

If you're developing or interacting with traditional Ethereum contracts, it's recommended to use one of our favorite Ethereum libraries like ethers.js or web3.js.

CHAPTER 1

Getting Started

To install the client

1.1 Node.js

```
npm install @oasislabs/client
```

1.2 Browser

The Oasis CDN hosts the latest version of the library. It can be included in your HTML as follows:

```
<script src="https://cdn.oasiscloud.io/oasis-client-latest/client/index.browser.umd.js
↪"
        charset="utf-8"
        type="text/javascript">
</script>
```

CHAPTER 2

Examples

Here we walk through a workflow demonstrating the core apis provided by the client. These examples assume an oasis-rs service is being used.

2.1 Set the gateway

First, one must select a gateway, for example, the Oasis Gateway.

```
const oasis = require('@oasislabs/client');

// Create a gateway at the given url.
const gateway = new oasis.gateways.Gateway('https://gateway.devnet.oasiscloud.io')

// Connect the library to the gateway.
oasis.setGateway(gateway);
```

2.2 Deploy

After connecting, one can deploy a new service.

```
// Service bytecode read directly from a .wasm file compiled with `oasis build`.
const bytecode = require('fs').readFileSync('/path/to/target/service/my-service.wasm
↪');

// Service constructor args.
const arg = "this is an argument";

// Deploy it through the connected gateway.
const service = await oasis.deploy(arg, {
   bytecode,
});
```

2.3 Service

Alternatively, one can connect to a previously deployed Service.

```
// On-chain address of the service (dummy address used here);.
const address = new oasis.Address('0x288e7e1cc60962f40d4d782950470e3705c5acf4');

// Connect to the service.
const service = await oasis.Service.at(address);
```

2.4 RPC

Once you've connected to a service, either by deploying or by connecting to an existing service, one can execute function calls on that service.

To make an rpc to a service

```
const returnValue = await service.myMethod();
```

2.5 Event Listeners

In addition to rpcs, one can register event listeners.

```
service.addEventListener('MyEvent', (event) => {
  // do something...
});
```

2.6 Wallets and Web3 Gateways

In the examples above, we've used a Gateway to pay for and sign transactions. This is useful when you want the client to operate without a wallet, but sometimes you want more control. In such cases, it's suggested to use a wallet and web3 gateway which will allow the client to sign and send raw transactions.

```
const oasis = require('@oasislabs/client');

// Wallet private key.
const privateKey = '0x1ad288d73cd2fff6ecf0a5bf167f59e9944559cd70f66cb70170702a0b4f3bd5
↪';

// Wallet for signing and paying for transactions.
const wallet = new oasis.Wallet(privateKey);

// Etheruem gateway responsible for signing transactions.
const gateway = new oasis.gateways.Web3Gateway('wss://web3.devnet.oasiscloud.io/ws',␣
↪wallet);

// Configure the gateway to use.
oasis.setGateway(gateway);
```

2.7 Web3 Options

When using a wallet and web3 gateway, one can also specify the options for the transaction. This is especially useful when working with confidential services, because the gasLimit *must* be explicitly supplied (estimate gas isn't provided for confidential services).

Note that the web3 options must always be the *last* argument given to an rpc method, after all rpc specific arguments.

```
service.myMethod({
  gasLimit: '0xf00000',
});
```

CHAPTER 3

setGateway

The `oasis.setGateway` method configures the client so that all service communication is done through the given *OasisGateway*. This method should be called before interacting with any services.

3.1 setGateway

```
oasis.setGateway(gateway)
```

3.1.1 Parameters

1. `gateway` - *OasisGateway*: The gateway to facilitate all service communications.

CHAPTER 4

deploy

The `oasis.deploy` method deploys a service via an Oasis gateway.

4.1 deploy

```
deploy(...args, options)
```

4.1.1 Parameters

1. `args` - The positional constructor arguments for the service. Note that the type and number of these arguments depend on the service being constructed.
2. `options` - `Object` (optional): The service deploy options. Must be the last argument in the deploy function call.

4.1.2 options

- `bytecode` - `string | Uint8Array`: The bytecode for the service.
- `header` - `Object` (optional): The deploy header. See the default values below.
- `gateway` - *OasisGateway* (optional): The client backend to communicate with an oasis gateway.
- `gasLimit` - `string` | `number` (optional): Gas limit to use for the transaction.
- `value` - `string` | `number` (optional): Value to send in the transaction.

4.1.3 header

- `confidential` - `boolean` (optional): True if the service should be confidential. Defaults to true.

- `expiry` - `number` (optional): Unix timestamp defining when the service expirs. Defaults to 100 years from the current timestamp.

4.1.4 Returns

Service: The deployed service instance with all rpc endpoints attached.

CHAPTER 5

Service

`oasis.Service` represents a service deployed on the Oasis network. Services deployed to the local chain can only interact with other locally-deployed services; integration tests will want to mock third-party services.

To construct a service:

5.1 Service

```
new oasis.Service(idl, address [, options])
```

5.1.1 Parameters

1. `idl` - `Idl`: The json idl defining the service interface
2. `address` - `string | Uint8Array`: The address of the service.
3. `options` - `Object` (optional):

5.1.2 options

- `gateway` - *OasisGateway* (optional): The client backend to communicate with an oasis gateway
- `db` - `Db` (optional): The persistent storage interface for the client's key manager.

5.1.3 Returns

`Service`: A service object with all rpc endpoints attached.

5.2 Service.at

A more convenient api to attach to a previously deployed service is the *Service.at* method, which will fetch the on-chain code and extract the idl automatciially.

Note: this method should only be with oasis-rs services.

```
await oasis.Service.at(address [, options])
```

5.2.1 Parameters

1. `address` - `string | Unit8Array`: The address of the service to connect to.
2. `options` - `Object` (optional): The service deploy options. See above.

5.2.2 Returns

`Promise<Service>`: A promise resolving to a service object with all rpc endpoints attached.

5.2.3 Rpc Methods

To make rpc requests to a service, invoke the rpc directly on the service object. For example,

```
// Deployed service address.
const address = ...;

// Connects to the remote service.
const service = await Service.at(address);

// Service specific rpc parameter.
const argument = 'this is an argument to a Service rpc';

// RpcOptions.
const options = { gasLimit: '0xe79732' };

// Makes an rpc request to `myMethod` and returns the result.
const returnValue = await service.myMethod(argument, options);
```

Note: The client will ensure all Service api methods are camelCase, as is idiomatic JavaScript, even if your on-chain service uses snake_case, as is idiomatic Rust.

The positional arguments for a given rpc should be passed directly into the method. In addition, one can **optionally** specify `RpcOptions`. When used, these options must be the last argument given to a method.

Important: Confidential Services

When making RPCs to **confidential** services, one **must** specify the `gasLimit` option. The client can't estimiate the gas limit when state is confidential.

A Service's method call returns only after the transaction has been finalized by the Oasis network.

RpcOptions

- `gasLimit` - `string` | `number` (optional): Gas limit to use for the transaction.
- `value` - `string` | `number` (optional): Value to send in the transaction.
- `aad` - `string` (optional): Additional authenticated data exposed to the confidential runtime.

5.2.4 addEventListener

To listen to events emitted by the service, use the `addEventListener` method.

```
service.addEventListener(event, function listener(event) {
  console.log('Received the event, ' event);
});
```

An event is emitted only after the block containing it has been finalized by the Oasis network.

Note: In the same way rpc method names are camelCase, event object keys are camelCase, even if the corresponding service is snake_case.

Parameters

1. `event` - `String`: The name of the event.
2. `listener` - `Listener`: A function taking a single event as a parameter.

5.2.5 removeEventListener

To stop listening to events emitted by the service, use the `removeEventListener` method. It's suggested to use this method to properly cleanup gateway subscriptions that result from creating event listeners.

```
service.removeEventListener(event, listener);
```

Parameters

1. `event` - `String`: The name of the event.
2. `listener` - `Listener`: The listener function previously given to `addEventListener`.

CHAPTER 6

gateways

The `oasis.gateways` namespace provides objects implementing the `OasisGateway` interface, defining the client's backend used to send transactions to the Oasis network. It should be rare to need to interact with this interface; however, the definition is provided for those who'd like to implement it.

6.1 interface OasisGateway

The interface is defined with the following TypeScript source.

The following two implementations are provided.

6.2 Gateway

`oasis.gateways.Gateway` provides an implementation of `OasisGateway` that communicates with a developer-gateway.

It currently only supports HTTP.

```
new oasis.gateways.Gateway(url, httpHeaders);
```

6.2.1 Parameters

1. `url` - `String`: The url of the gateway.
2. `httpHeaders` - `Object`: The http headers to use for authentiication with the gateway. For example, `{ headers: new Map([['X-OASIS-INSECURE-AUTH', 'VALUE']]) }`.

Warning: The `oasis.gateways.Gateway` server is not yet readily availiable in public Oasis infrastructure. For now, it's recommended to use `oasis.gateways.Web3Gateway`.

6.3 Web3Gateway

`oasis.gateways.Web3Gateway` provides an implementation of `OasisGateway` that communicates with a web3 gateway. It's suggested to use this package if you want the client to sign transactions with its own wallet and specify transaction options like gas price and gas limit.

It currently only supports WebSockets and can be instantiated as follows:

```
let url = 'wss://web3.devnet.oasiscloud.io/ws';
let wallet = new oasis.Wallet(SECRET_KEY);

new oasis.gateways.Web3Gateway(url, wallet);
```

6.3.1 Parameters

1. `url` - `String`: The url of the gateway.
2. `wallet` - `Wallet`: The wallet to sign transactions. For convenience, we package and suggest using the ethers.js wallet.

6.3.2 Methods

In addition to implementing the `OasisGateway` interface, the `Web3Gateway` exposes a subset of the Web3 JSON RPC spec along with Oasis extensions.

Supported namespaces are

- `eth_`
- `oasis_`
- `net_`

Examples

To execute a web3 rpc method, simply access the namespace followed by the method.

For example, to retrieve the latest block using `eth_getBlockByNumber`,

```
await gateway.eth.getBlockByNumber('latest', true);
```

To get the expiration of a given service using `oasis_getExpiry`,

```
await gateway.oasis.getExpiry(address);
```

eth.protocolVersion

Parameters

none

Returns

`String` - The current ethereum protocol version.

eth.gasPrice

Parameters

none

Returns

`String` - hex string representing the current gas price in wei.

eth.blockNumber

Parameters

none

Returns

`String` - hex string repreenting the current block number the gateway is on.

eth.getBalance

Parameters

1. `String` - hex string 20 byte address to check for balance.
2. `String` - hex string block number, or the string “latest”, “earliest” or “pending” (defaults to “latest”).

Returns

`String` - hex string the current balance in wei.

eth.getStorageAt

Parameters

1. `String` - hex string 20 byte address of the storage.
2. `String` - hex string of the position in the storage.
3. `String` - hex string of the block number, or the string “latest”, “earliest” or “pending” (defaults to “latest”).

Returns

`String` - hex string of the value at this storage position for the given address.

eth.getTransactionCount

Parameters

1. `String` - hex string of the address to get the transaction count for.
2. `String` - hex string of the block number, or the string "latest", "earliest" or "pending" (defaults to "latest").

Returns

`String` - hex string of the number of transactions send from this address.

eth.getBlockTransactionCountByHash

Parameters

1. `String` - hex string hash of a block.

Returns

`String` - hex string of the number of transactions in this block.

eth.getBlockTransactionCountByNumber

Parameters

1. `String` - hex string of a block number, or the string "earliest", "latest" or "pending" (defaults to "latest").

Returns

`String` - hex string of the number of transactions in this block.

eth.getCode

Parameters

1. `String` - hex string of the address.
2. `String` - hex string of the block number, or the string "latest", "earliest" or "pending" (defaults to "latest').

Returns

`String` - hex string of the code of the given address.

eth.sendTransaction

In order to use this method, the `Web3Gateway` must be created with a `Wallet`. Note that this is slightly different from the canonical web3 eth_sendTransaction in that the client will sign the transaction and convert it into an eth_sendRawTransaction before sending it to the remote gateway.

Parameters

1. Object - The transaction object

- to: `String` - (optional when creating new contract) The address the transaction is directed to.
- gas: `String` - (optional, default: 90000) Integer of the gas provided for the transaction execution. It will return unused gas.
- gasPrice: `String` - (optional) Integer of the gasPrice used for each paid gas

- value: `String` - (optional) Integer of the value sent with this transaction
- data: `String` - (optional) The data field of the transaction.
- nonce: `String` - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

Returns `String` - 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 validated, when you created a contract.

eth.sendRawTransaction

Parameters

. *''String'* - The signed transaction data.

Returns

`String` - hex string of the 32 byte 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 validated, when you created a contract.

eth.call

Executes a transaction at the gateway without creating a transaction on the block chain. Note that this feature is not available for confidential contracts, because Web3 gateways don't have access to confidential state.

Parameters

1. Object - The transaction call object

- from: `String` - (optional) The address the transaction is sent from.
- to: `String` - when creating new contract) The address the transaction is directed to.
- gas: `String` - (optional, default: 90000) Integer of the gas provided for the transaction execution. It will return unused gas.
- gasPrice: `String` - (optional) Integer of the gasPrice used for each paid gas
- value: `String` - (optional) Integer of the value sent with this transaction
- data: `String` - (optional) The data field of the transaction.
- nonce: `String` - (optional) Integer of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

Returns

`String` - hex string of the return value of the transaction

eth.estimateGas

Estimates gas for a given transaction by executing a transaction at the gateway and recording the gas used. The transaction is not added to the blockchain and so doesnt not affect state. Note that this feature is not available for confidential contracts, because Web3 gateways don't have access to confidential state.

Parameters

1. Object - The transaction call object

- from: `String` - (optional) The address the transaction is sent from.
- to: `String` - (optional) when creating new contract) The address the transaction is directed to.
- gas: `String` - (optional) hex string of the gas provided for the transaction execution. It will return unused gas.
- gasPrice: `String` - (optional) hex string of the gasPrice used for each paid gas
- value: `String` - (optional) hex string of the value sent with this transaction
- data: `String` - (optional) the data field of the transaction.
- nonce: `String` - (optional) hex stirng of a nonce. This allows to overwrite your own pending transactions that use the same nonce.

Returns

`String` - hex string of the amount of gas used for executing a transaction at the gateway.

eth.getBlockByHash

Parameters

1. `String` - hex string hash of a block.
2. `Boolean` - if true it returns the full transaction objects, if false only the hashes of the transactions.

Returns

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

- number: `String` (hex) - the block number. null when its pending block.
- hash: `String` (hex), 32 Bytes - hash of the block. null when its pending block.
- parentHash: `String` (hex), 32 Bytes - hash of the parent block.
- nonce: `String` (hex), 8 Bytes - hash of the generated proof-of-work. null when its pending block.
- sha3Uncles: `String` (hex), 32 Bytes - SHA3 of the uncles data in the block.
- logsBloom: `String` (hex), 256 Bytes - the bloom filter for the logs of the block. null when its pending block.
- transactionsRoot: `String` (hex), 32 Bytes - the root of the transaction trie of the block.
- stateRoot: `String` (hex), 32 Bytes - the root of the final state trie of the block.
- receiptsRoot: `String` (hex), 32 Bytes - the root of the receipts trie of the block.
- miner: `String` (hex), 20 Bytes - the address of the beneficiary to whom the mining rewards were given.
- difficulty: `String` (hex) - integer of the difficulty for this block.
- totalDifficulty: `String` (hex) - integer of the total difficulty of the chain until this block.
- extraData: `String` (hex) - the “extra data” field of this block.
- size: `String` (hex) - integer the size of this block in bytes.
- gasLimit: `String` (hex) - the maximum gas allowed in this block.
- gasUsed: `String` (hex) - the total used gas by all transactions in this block.
- timestamp: `String` (hex) - 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.

eth.getBlockByNumber

Parameters

1. `String` - 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.

Returns

See *eth.getBlockByHash*.

eth.getTransactionByHash

Parameters

1. `String` (hex), 32 Bytes - hash of a transaction

Returns

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

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

eth.getTransactionByBlockHashAndIndex

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

Parameters

1. `String` (hex), 32 Bytes - hash of a block.
2. `String` (hex) - integer of the transaction index position.

Returns

See *eth.getTransactionByHash*.

eth.getTransactionByBlockNumberAndIndex

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

Parameters

1. `String` (hex) - a block number, or the string "earliest", "latest" or "pending" (defaults to "latest").
2. `String` (hex) - the transaction index position.

Returns

See *eth.getTransactionByHash*.

eth.getTransactionReceipt

Returns the receipt of a transaction by transaction hash.

Parameters

1. `String` (hex), 32 Bytes - hash of a transaction

Returns

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

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

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

Object - The filter options:

- fromBlock: `String` (hex) - (optional, default: “latest”) Integer block number, or “latest” for the last mined block or “pending”, “earliest” for not yet mined transactions.
- toBlock: `String` (hex) - (optional, default: “latest”) Integer block number, or “latest” for the last mined block or “pending”, “earliest” for not yet mined transactions.
- address: `String` (hex) | Array, 20 Bytes - (optional) Contract address or a list of addresses from which logs should originate.
- topics: Array of `String` (hex), - (optional) Array of 32 Bytes `String` (hex) topics. Topics are order-dependent. Each topic can also be an array of `String` (hex) with “or” options.

Returns

`String` - A filter id.

Example

```
gateway.eth.newFilter({
  "fromBlock": "0x1",
  "toBlock": "0x2",
  "address": "0x8888f1f195afa192cfee860698584c030f4c9db1",
  "topics": ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b",␣
↪null, ["0x000000000000000000000000a94f5374fce5edbc8e2a8697c15331677e6ebf0b",
↪"0x000000000000000000000000aff3454fce5edbc8cca8697c15331677e6ebccc"]]
})
```

eth.newBlockFilter

Creates a filter in the gateway, to notify when a new block arrives. To check if the state has changed, call *eth.getFilterChanges*.

Parameters

None

Returns

`String` - A filter id.

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

`String` - A filter id.

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. `String` (hex) - The filter id.

Returns

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

eth.getFilterChanges

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

Parameters

1. `String` (hex) - the filter id.

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 (`String` (hex), 32 Bytes), e.g. `["0x3454645634534..."]`.
- For filters created with eth_newPendingTransactionFilter the return are transaction hashes (`String` (hex), 32 Bytes), e.g. `["0x6345343454645..."]`.
- For filters created with eth_newFilter logs are objects with following params:
  - removed: `boolean` - true when the log was removed, due to a chain reorganization. false if its a valid log.
  - logIndex: `String` (hex) - integer of the log index position in the block. null when its pending log.
  - transactionIndex: `String` (hex) - integer of the transactions index position log was created from. null when its pending log.
  - transactionHash: `String` (hex), 32 Bytes - hash of the transactions this log was created from. null when its pending log.
  - blockHash: `String` (hex), 32 Bytes - hash of the block where this log was in. null when its pending. null when its pending log.
  - blockNumber: `String` (hex) - the block number where this log was in. null when its pending. null when its pending log.
  - address: `String` (hex), 20 Bytes - address from which this log originated.
  - data: `String` (hex) - contains the non-indexed arguments of the log.
  - topics: Array of `String` (hex) - Array of 0 to 4 32 Bytes `String` (hex) 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.)

eth.getFilterLogs

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

Parameters

1. `String` (hex) - The filter id.

Returns

See *eth_getFilterChanges*.

eth.getLogs

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

Parameters

`Object` - The filter options:

- fromBlock: `String` - (optional, default: "latest") Integer block number, or "latest" for the last mined block or "pending", "earliest" for not yet mined transactions.
- toBlock: `String` - (optional, default: "latest") Integer block number, or "latest" for the last mined block or "pending", "earliest" for not yet mined transactions.
- address: `String` (hex) | Array, 20 Bytes - (optional) Contract address or a list of addresses from which logs should originate.
- topics: Array of `String` (hex), - (optional) Array of 32 Bytes `String` (hex) topics. Topics are order-dependent. Each topic can also be an array of `String` (hex) with "or" options.
- blockhash: `String` (hex), 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.

Returns

See *eth_getFilterChanges*.

net.version

Parameters

none

Returns

`String` - the current network id.

- "1": Ethereum Mainnet
- "2": Morden Testnet (deprecated)
- "3": Ropsten Testnet
- "4": Rinkeby Testnet
- "42": Kovan Testnet
- ...
- "42261": Oasis Devnet

net.listening

Parameters

none

Returns

`Boolean` - true when the gateway is listening for network connections. Otherwise false.

oasi.getExpiry

Parameters

1. Address: 0x prefixed 20-byte hex string representing the address of the service.

Returns

the expiration timestamp for the service.

oasis.getPublicKey

Parameters

1. Address: 0x prefixed 20-byte hex string representing the address of the service for which to retrieve the public key.

Returns

the public key created by the Key Manager, used for encrypted communication with the service.

6.3.3 Subscriptions

To make a web3 subscription use the `eth_subscribe` method. Unlike other RPCs, instead of returning the server response, `eth_subscribe` will resolve to an `EventEmitter` object, emitting events on the `data` topic.

For example, to subscribe to logs,

```
const subscription = await gateway.eth.subscribe('logs', {
  address: '0x...'
  topics: ['0x...']
});

subscription.on('data', (event) => {
  // Do something with your event.
});
```

To subscribe to new block headers,

```
const subscription = await gateway.eth.subscribe('newHeads');

subscription.on('data', (event) => {
  // Do something with your block headers.
});
```

To unsubscribe, give the subscription id to the `eth_unsubscribe` method. Continuing the above example,

```
await gateway.eth.unsubscribe(subscription.id);
```

CHAPTER 7

Wallet

`oasis.Wallet` provides an instance of `ethers.Wallet`. See the ethers docs.

CHAPTER 8

workspace

This feature is for node.js only.

The `oasis.workspace` namespace provides easy access to the build artifacts in your local project workspace for **deployment**. This is particularly useful when testing and is meant to be used when developing oasis-rs services with oasis-cli, as it relies on the oasis-cli configuration file.

8.1 Examples

Suppose we have the following service:

```
#[derive(Service)]
pub struct MyService;

impl MyService {
    pub fn new(_ctx: &Context, arg1: u64, arg2: u64) -> Result<Self, String> {
        Ok(Self {})
    }
}
```

8.2 Deploying a service

Then you can deploy it with `oasis.workspace` as follows:

```
oasis.workspace.MyService.deploy(arg1, arg2, options);
```

Where `arg1` and `arg2` refer to the positional constructor arguments of `MyService` and `options` is an (optional) instance of *RpcOptions*.

Note that the number of arguments will vary per service and the `options` must be the **last** argument given.

8.3 Using a gateway

You can also fetch the gateway specified in your configuration file and set that as your default gateway. For example,

```
// Build the gateway object from the workspace configuration file.
const gateway = await oasis.workspace.gateway();

// Set the client's default gateway.
oasis.setGateway(gateway);
```

8.4 Configuration

The workspace can be configured using a config.toml file to define the gateway and an optional mnemonic or private key. For example,

```
[profile.default]
endpoint = 'https://gateway.devnet.oasiscloud.io'
private_key = ''

[profile.local]
endpoint = 'ws://localhost:8546'
mnemonic = 'range drive remove bleak mule satisfy mandate east lion minimum unfold␣
↪ready'
```

8.5 Environment Variables

Although not recommended (because they are meant to be configured with oasis-cli), one can also override the following environment variables.

- `OASIS_WORKSPACE` - Path to the workspace root directory. Defaults to the root directory of the local git repository.
- `OASIS_PROFILE` - The config profile to use. Defaults to `default`.
- **`OASIS_CONFIG` - Path to the config file to use. Defaults to a system specific path, e.g.**
  - macOS and Linux - `~/.config/oasis/config.toml`

8.6 How it works

When `oasis.workspace` is first accessed, e.g., via `oasis.workspace.MyService` the client lazily populates the namespace by searching for `target/service/*.wasm` files in your local git repository directory subtree, constructing and attaching the found service definitions to the namespace.

For an example workspace, see the template.

CHAPTER 9

disconnect

The `oasis.disconnect` method disconnects the client from the default *OasisGateway*, set via *oasis.setGateway*. When using websockets, this method should be called to clean up any outstanding connections, e.g., when testing.

9.1 disconnect

```
oasis.disconnect()
```

CHAPTER 10

utils

`oasis.utils` provides a collection of client utilities.

10.1 utils.encrypt

```
utils.encrypt(
  nonce,
  plaintext,
  peerPublicKey,
  privateKey,
  aad
);
```

10.1.1 Parameters

- `nonce` - `Uint8Array`: The nonce used to encrypt the ciphertext.
- `plaintext` - `Uint8Array`: The text to be encrypted
- `peerPublicKey` - `Uint8Array`: The public key to which the ciphertext will be encrypted.
- `publicKey` - `Uint8Array`: The public key of the entity encrypting the data.
- `privateKey` - `Uint7Array`: The private key of the entity encrypting the data.
- `aad` - `Uint8Array` the additional authenticated data for the AEAD.

10.1.2 Returns

`Uint8Array`: The encoded wire format of the ciphertext `PUBLIC_KEY || CIPHER_LENGTH || AAD_LENGTH || CIPHER ||AAD || NONCE`, where `CIPHER_LENGTH` AND `AAD_LENGTH` are encoded as big endian uint64.

10.2 utils.decrypt

Decrypts the given ciphertext using Deoxysii.js and the wire format specified above.

```
utils.decrypt(encryption, secretKey);
```

10.2.1 Parameters

1. `encryption` - `Uint8Array`: The encrypted data in the wire formated specified above.
2. `secretKey` - `Uint8Array`: The secret key to which the data was encrypted.

10.2.2 Returns

`Object`: The decryption of the object with a key for each component of the decryption.

- `nonce` - `Uint8Array`: The nonce used to encrypt the ciphertext.
- `plaintext` - `Uint8Array`: The decrypted ciphertext.
- `peerPublicKey` - `Uint8Array`: The public key from which the ciphertext encrypted.
- `aad` - `Uint8Array` the additional authenticated data for the AEAD.

10.3 utils.header.parseFromCode

```
oasis.utils.header.parseHex(deploycode);
```

10.3.1 Parameters

1. `deploycode` - `Uint8Array | String`: The deployed bytecode of a service, prefixed with the header wire format:

```
b'\0sis' || version (2 bytes big endian) || length (2 bytes big endian) || json-header
```

10.3.2 Returns

`Object`: The deploy header with all fields.

- `version` - `number`: The version number of the header.
- `expirty` - `number`: Service expiry timestamp
- `confidential` - `boolean`: True if the service is confidential.

10.4 utils.bytes.parseHex

```
oasis.utils.bytes.parseHex(hexStr);
```

10.4.1 Parameters

1. `hexStr` - `String`: Hex string to parse.

10.4.2 Returns

`Uint8Array`: Transformed representation the given hex string.

10.5 utils.bytes.toHex

```
oasis.utils.bytes.toHex(byteArray);
```

10.5.1 Parameters

1. `byteArray` - `Uint8Array`: Byte array to convert into a hex string

10.5.2 Returns

`String`: Transformed representation of the given byte array.

10.6 utils.bytes.encodeUtf8

```
oasis.utils.header.encodeUtf8(input);
```

10.6.1 Parameters

1. `input` - `string`: String to encode into a utf8 encoded byte array

10.6.2 Returns

`Uint8Array`: Utf8 encoded byte earray

10.7 utils.bytes.decodeUtf8

```
oasis.utils.header.decodeUtf8(input);
```

10.7.1 Parameters

1. `input` - `Uint8Array`: Byte array previously encoded with `utils.bytes.encodeUtf8`

10.7.2 Returns

`String`: Utf8 encoded string

10.8 utils.cbor.encode

```
oasis.utils.cbor.encode(input);
```

10.8.1 Parameters

1. `input` - `Object`: JSON object to cbor encode

10.8.2 Returns

`Uint8Array`: Cbor encoded byte array

10.9 utils.cbor.decode

```
oasis.utils.cbor.decode(input);
```

10.9.1 Parameters

1. `input` - `Uint8Array`: Cbor encoded byte array

10.9.2 Returns

`Object`: Decoded JSON object

10.10 utils.keccak256

Keccak256 implementation from js-sha3.

10.11 utils.idl.fromWasm

```
oasis.utils.idl.fromWasm(bytecode);
```

10.11.1 Parameters

1. `bytecode` - `Uint8Array`: Raw wasi bytecode compiled with `oasis build`. See the oasis-cli.

10.11.2 Returns

`Promise<Object>`: Promise resolving to the Idl extracted from the *oasis-interface* section of the given bytecode.

10.12 utils.idl.fromWasmSync

```
oasis.utils.idl.fromWasmSync(input);
```

A synchronous version of `utils.idl.fromWasm`.