Hyperspace Remote Procedure Calls

Remote Procedure Calls, or RPCs, are a way for an external program (eg. a frontend) to communicate with a node. They are used for checking storage values, submitting transactions, and querying the current consensus authorities with any client that speaks json RPC. One widely available option for using RPC is curl.
Example:

```
#!/bin/bash
"curl -H "Content-Type: application/json" -d '{"id":"1", "jsonrpc":"2.0",
"method":"state_getRuntimeVersion", "params":[]}' https://vm.mvs.org/mainnet_rpc"
```

List of calls:

* calls marked with an asterix are under development

account_nextIndex (*account: AccountId)*
Returns the next valid index (aka nonce) for given account.
This method takes into consideration all pending transactions currently
in the pool and if no transactions are found in the pool it fallbacks
to query the index from the runtime (aka. state nonce).

author_hasKey (*public_key: Bytes, key_type: String*)
Checks if the keystore has private keys for the given public key and
key type. Returns `true` if a private key could be found.

author_hasSessionKeys (*session_keys: Bytes*)
Checks if the keystore has private keys for the given session public
keys. `session_keys` is the SCALE encoded session keys object from the
runtime. Returns `true` iff all private keys could be found.

author_insertKey (*key_type: String,suri: String, public: Bytes*)
Insert a key into the keystore.

author_pendingExtrinsics ()
Returns all pending extrinsics, potentially grouped by sender.

author_removeExtrinsic (*bytes_or_hash: Vec<hash::ExtrinsicOrHash<Hash>>*)
Remove given extrinsic from the pool and temporarily ban it to prevent
reimporting.

author_rotateKeys ()
Generate new session keys and returns the corresponding public keys.

author_submitAndWatchExtrinsic (*metadata: Self::Metadata,*
subscriber: Subscriber<TransactionStatus<Hash, BlockHash>>,
bytes: Bytes)

Submit an extrinsic to watch. See [`TransactionStatus`](sp_transaction_pool::TransactionStatus) for details on transaction life cycle.

author_submitExtrinsic (*extrinsic: Bytes*)
Submit hex-encoded extrinsic for inclusion in block.

author_unwatchExtrinsic (*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
Unsubscribe from extrinsic watching.

babe_epochAuthorship ()
Returns data about which slots (primary or secondary) can be claimed in the current epoch with the keys in the keystore.

balances_usableBalance (*instance: u8,*
who: AccountId)
Node-specific RPC methods for interaction with balances.

chain_getBlock (*hash: Option<Hash>*)
Get header and body of a relay chain block.

chain_getBlockHash (*hash: Option<ListOrValue<NumberOrHex>>*)
Get hash of the n-th block in the canon chain. By default returns latest block hash.

chain_getFinalisedHead ()
Get hash of the last finalized block in the canon chain.

chain_getHead (*hash: Option<ListOrValue<NumberOrHex>>*)
Get hash of the n-th block in the canon chain. By default returns latest block hash.

chain_getHeader (*hash: Option<Hash>*)
Get header of a relay chain block.

chain_getRuntimeVersion (*hash: Option<Hash>*)
Get the runtime version.

chain_subscribeAllHeads (*metadata: Self::Metadata,*
subscriber: Subscriber<Header>)
Finalized head subscription operations.

All head subscription (*metadata: Self::Metadata,*
subscriber: Subscriber<Header>)
Finalized head subscription operations.

chain_subscribeNewHead (*metadata: Self::Metadata,*
subscriber: Subscriber<Header>)
Finalized head subscription operations.

chain_subscribeNewHeads (*metadata: Self::Metadata,*
subscriber: Subscriber<Header>)
Finalized head subscription operations.

chain_subscribeRuntimeVersion (*metadata: Self::Metadata,*
subscriber: Subscriber<RuntimeVersion>)

New runtime version subscription alias of state_subscribeRuntimeVersion

chain_unsubscribeAllHeads* (metadata: Option<Self::Metadata>,*
*id: SubscriptionId***)**

chain_unsubscribeFinalisedHeads() alias chain_unsubscribeAllHeads

chain_unsubscribeNewHead alias chain_unsubscribeAllHeads

chain_unsubscribeNewHeads (metadata: Option<Self::Metadata>,*
id: SubscriptionId***)***

chain_unsubscribeRuntimeVersion(metadata: Option<Self::Metadata>,*
*id: SubscriptionId****)***
Finalized head and RuntimeVersion unsubscription operations.

childstate_getKeys (*child_storage_key: PrefixedStorageKey,*
prefix: StorageKey,
hash: Option<Hash>)
Returns the keys with prefix from a child storage, leave empty to get
all the keys

childstate_getStorage (*child_storage_key: PrefixedStorageKey,*
key: StorageKey,
hash: Option<Hash>)
Returns a child storage entry at a specific block's state

childstate_getStorageSize (*child_storage_key: PrefixedStorageKey,*
key: StorageKey,
hash: Option<Hash>)
Returns the size of a child storage entry at a block's state.

eth_accounts ()*
Returns EVM accounts list.

eth_blockNumber ()
Returns highest block number from EVM perspective.

eth_call (*_: CallRequest,*
_: Option<BlockNumber>)
Call contract, returning the output data.

eth_chainId ()
Returns the chain ID used for transaction signing at the current best block. None is returned if not.

eth_estimateGas (*_: CallRequest,*
_: Option<BlockNumber>)
Estimate gas needed for execution of given contract.

eth_gasPrice ()
Returns current gas_price.

eth_getBalance (*_: H160,*
_: Option<BlockNumber>)
Returns balance of the given account.

eth_getBlockByHash (*_: H256,*
_: bool)
Returns block with given hash.

eth_getBlockByNumber (*_: BlockNumber,*
_: bool)
Returns block with given number.

eth_getBlockTransactionCountByHash (*_: H256*)
Returns the number of transactions in a block with given hash.

eth_getBlockTransactionCountByNumber (*_: BlockNumber*)
Returns the number of transactions in a block with given block number.

eth_getCode (*_: H160,*
_: Option<BlockNumber>)
Returns the code at given address at given time (block number).

eth_getLogs (*_: Filter*)
Returns logs matching given filter object.

eth_getStorageAt (*_: H160,*
_: U256,
_: Option<BlockNumber>)
Returns content of the storage at given address.

eth_getTransactionByBlockHashAndIndex (*_: H256,*
_: Index)
Returns transaction at given block hash and index.

eth_getTransactionByBlockNumberAndIndex(*v_: BlockNumber,*
_: Index)
Returns transaction by given block number and index.

eth_getTransactionByHash (*_: H256*)
Get transaction by its hash.

eth_getTransactionCount (*_: H160,*
_: Option<BlockNumber>)
Returns the number of transactions sent from given address at given time (block number).

eth_getTransactionReceipt (*_: H256*)
Returns transaction receipt by transaction hash.

eth_getUncleByBlockHashAndIndex (*_: H256,*
_: Index)
Returns Unlce by block hash and index

eth_getUncleByBlockNumberAndIndex (*_: BlockNumber,*
_: Index)
Returns an uncle at given block and index.

eth_getUncleCountByBlockHash (*_: H256*)
Returns the number of uncles in a block with given hash.

eth_getUncleCountByBlockNumber (*_: BlockNumber*)
Returns the number of uncles in a block with given block number.

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

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

eth_mining ()
Returns true if client is actively mining new blocks.

eth_protocolVersion ()
Returns protocol version encoded as a string (quotes are necessary here).

eth_sendRawTransaction (*_: Bytes*)
Sends signed transaction, returning its hash.

eth_sendTransaction (*_: TransactionRequest*)
Sends transaction; will block waiting for signer to return the transaction hash.

eth_submitHashrate (*_: U256,*
_: H256)
Used for submitting mining hashrate.

eth_submitWork (*_: H64,*
_: H256,
_: H256)
Used for submitting a proof-of-work solution.

eth_subscribe (*_: Self::Metadata,*
_: typed::Subscriber<pubsub::Result>,
_: pubsub::Kind,
_: Option<pubsub::Params>)
Subscribe to Eth subscription.

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

eth_unsubscribe (*_: Option<Self::Metadata>,*
_: SubscriptionId)
Unsubscribe from existing Eth subscription.

grandpa_proveFinality (*begin: Hash,*
end: Hash,
authorities_set_id: u64)
Prove finality for the given block number by returning the justification for the last block in the set and all the intermediary headers to link them together.

grandpa_roundState ()
Returns the state of the current best round state as well as the ongoing background rounds.

grandpa_subscribeJustifications (*metadata: Self::Metadata,*
subscriber: Subscriber<Notification>)
Returns the block most recently finalized by Grandpa, alongside side its justification.

grandpa_unsubscribeJustifications (*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
Unsubscribe from receiving notifications about recently finalized blocks.

headerMMR_genProof (*block_number_of_member_leaf: u64,*
block_number_of_last_leaf: u64)
Get the MMR proof for a certain height, block number of member leaf, block number of the lastest leafnet_listening,
Returns true if client is actively listening for network connections. Otherwise false.

net_peerCount ()
Returns number of peers connected to node.

net_version ()
Returns protocol version.

offchain_localStorageGet (*kind: StorageKind,*
key: Bytes)
Get offchain local storage under given key and prefix.

offchain_localStorageSet (*kind: StorageKind,*
key: Bytes,
value: Bytes)
Set offchain local storage under given key and prefix.

payment_queryFeeDetails (*encoded_xt: Bytes,*
at: Option<BlockHash>)
Query the fee of a payment

payment_queryInfo (*encoded_xt: Bytes,*
at: Option<BlockHash>)
Get details regarding payment fee.

staking_powerOf (*who: AccountId*)
Retrunt the power on a certain AccountId in a staking context.

state_call (*name: String,*
bytes: Bytes,
hash: Option<Hash>)
Call a contract's block state

state_callAt (*name: String,*
bytes: Bytes,
hash: Option<Hash>)
Call a contract at a block's state.

state_getKeys (*prefix: StorageKey,*
hash: Option<Hash>)
Returns the keys with prefix, leave empty to get all the keys.

state_getKeysPaged (*prefix: Option<StorageKey>,*
count: u32,
start_key: Option<StorageKey>,
hash: Option<Hash>)
Returns the keys with prefix with pagination support.
Up to `count` keys will be returned.
If `start_key` is passed, return next keys in storage in lexicographic order.

state_getKeysPagedAt (*prefix: Option<StorageKey>,*
count: u32,
start_key: Option<StorageKey>,
hash: Option<Hash>)
Returns the keys with prefix with pagination support.
Up to `count` keys will be returned.
If `start_key` is passed, return next keys in storage in lexicographic order.

state_getMetadata ()
Returns the runtime metadata as an opaque blob.state_getPairs,
Returns the keys with prefix, leave empty to get all the keys

state_getReadProof (*keys: Vec<StorageKey>,*
hash: Option<Hash>)
Returns proof of storage entries at a specific block's state.

state_getRuntimeVersion (*hash: Option<Hash>*)
Get the runtime version.

state_getStorage (*key: StorageKey,*
hash: Option<Hash>)
Returns a storage entry at a specific block's state.

state_getStorageAt (*key: StorageKey,*
hash: Option<Hash>)
Returns the hash of a storage entry at a block's state.

state_getStorageHash (*key: StorageKey,*
hash: Option<Hash>)
Returns the hash of a storage entry at a block's state.

state_getStorageHashAt (*key: StorageKey,*
hash: Option<Hash>)
Returns the hash of a storage entry at a block's state.

state_getStorageSize (*key: StorageKey,*
hash: Option<Hash)
Returns the size of a storage entry at a block's state.

state_getStorageSizeAt (*key: StorageKey,*
hash: Option<Hash)

state_queryStorage (*keys: Vec<StorageKey>,*
block: Hash,
hash: Option<Hash>)
Query historical storage entries (by key) starting from a block given as the second parameter.
NOTE This first returned result contains the initial state of storage for all keys.
Subsequent values in the vector represent changes to the previous state (diffs).

state_queryStorageAt (*keys: Vec<StorageKey>,*
at: Option<Hash>)
Query storage entries (by key) starting at block hash given as the second parameter.

state_subscribeRuntimeVersion (*metadata: Self::Metadata,*
subscriber: Subscriber<RuntimeVersion)
New runtime version subscription

state_subscribeStorage (*metadata: Self::Metadata,*
subscriber: Subscriber<StorageChangeSet<Hash>>,
keys: Option<Vec<StorageKey>)
New storage subscription

state_unsubscribeRuntimeVersion (*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
Unsubscribe from runtime subscription

state_unsubscribeStorage (*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
Unsubscribe from storage subscription

subscribe_newHead (*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
New head subscription

sync_state_genSyncSpec (*raw: bool*)
Returns the json-serialized chainspec running the node, with a sync
state.

system_accountNextIndex (*account: AccountId*)
Returns the next valid index (aka nonce) for given account. This method
takes into consideration all pending transactions currently in the pool
and if no transactions are found in the pool it fallbacks to query the
index from the runtime (aka. state nonce).

system_addLogFilter (*directives: String*)
Adds the supplied directives to the current log filter. The syntax is
identical to the CLI `<target>=<level>`:`sync=debug,state=trace`

system_addReservedPeer (peer: String)
Adds a reserved peer. Returns the empty string or an error. The string
parameter should encode a `p2p` multiaddr.

system_chain ()
Get the chain's name. Given as a string identifier.

system_chainType ()
Get the chain's type.

system_dryRun (extrinsic: Bytes,
at: Option<BlockHash>)
Dry run an extrinsic at a given block. Return SCALE encoded ApplyExtrinsicResult.

system_dryRunAt (*extrinsic: Bytes,*
at: Option<BlockHash>)
Dry run an extrinsic at a given block. Return SCALE encoded ApplyExtrinsicResult

system_health ()
Return health status of the node. Node is considered healthy if it is:
- connected to some peers (unless running in dev mode)
- not performing a major sync

system_localListenAddresses ()
Returns the multiaddresses that the local node is listening on. The
addresses include a trailing `/p2p/` with the local PeerId, and are
thus suitable to be passed to `system_addReservedPeer` or as a bootnode
address for example.

system_localPeerId ()
Returns the base58-encoded PeerId of the node.

system_name ()
Get the node's implementation name. Plain old string.

system_networkState ()
Return the current network state

system_nodeRoles ()
Returns the roles the node is running as.

system_peers ()
Returns currently connected peers

system_properties ()
Get a custom set of properties as a JSON object, defined in the chain spec.

system_removeReservedPeer (*peer_id: String*)
Remove a reserved peer. Returns the empty string or an error. The string should encode only the PeerId.

system_resetLogFilter ()
Resets the log filter to defaults.

system_syncState()
Returns the state of the syncing of the node: starting block, current best block, highest known block.

system_version()
Get the node implementation's version. Should be a semver string.

unsubscribe_newHead (*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
Unsubscribe new head.

Unsubscribe from new head subscription(*metadata: Option<Self::Metadata>,*
id: SubscriptionId)
Unsubscribe from new head subscription.

web3_clientVersion ()
Returns current client version

web3_sha3 (*_: Bytes*)
Returns sha3 of the given data