# #

EVM

## #

Summary

This document defines the specification of theEthereum Virtual Machineopen in new windowas a Cosmos SDK module.

## #

Concept

### #

EVM

The Ethereum Virtual Machine (EVM) is a state machine that provides the necessary tools to run or create a contract on a given state.

### #

State DB

TheStateDB interface from geth represents an EVM database for full state querying of both contracts and accounts. The concrete type that fulfills this interface on Ethermint is theCommitStateDB .

### #

State Object

AstateObject represents an Ethereum account which is being modified. The usage pattern is as follows:

- First you need to obtain a state object.
- Account values can be accessed and modified through the object.

### #

Genesis State

Thex/evm moduleGenesisState defines the state necessary for initializing the chain from a previous exported height.

// GenesisState defines the evm module genesis state GenesisStatestruct { Accounts[ ] GenesisAccountjson:"accounts" TxsLogs[ ] TransactionLogsjson:"txs_logs" ChainConfig ChainConfigjson:"chain_config" Params Paramsjson:"params" } Genesis Accounts

TheGenesisAccount type corresponds to an adaptation of the EthereumGenesisAccount type. Its main difference is that the one on Ethermint uses a customStorage type that uses a slice instead of maps for the evmState (due to non-determinism), and that it doesn't contain the private key field.

It is also important to note that since theauth andbank SDK modules manage the accounts and balance state, theAddress must correspond to anEthAccount that is stored in theauth 's moduleAccountKeeper and the balance must match the balance of theEvmDenom token denomination defined on theGenesisState 'sParam . The values for the address and the balance amount maintain the same format as the ones from the SDK to make manual inspections easier on thegenesis.json .

// GenesisAccount defines an account to be initialized in the genesis state. // Its main difference between with Geth's GenesisAccount is that it uses a custom // storage type and that it doesn't contain the private key field. GenesisAccountstruct { Address ethcmn. Addressjson:"address" Balance* big. Intjson:"balance" Code hexutil. Bytesjson:"code,omitempty" Storage Storagejson:"storage,omitempty" } Transaction Logs

On every Ethermint transaction, its result contains the EthereumLog s from the state machine execution that are used by the JSON-RPC Web3 server for filter querying. Since Cosmos upgrades don't persist the transactions on the blockchain state, we need to persist the logs the EVM module state to prevent the queries from failing.

TxsLogs is the field that contains all the transaction logs that need to be persisted after an upgrade. It uses an array instead of a map to ensure determinism on the iteration.

// TransactionLogs define the logs generated from a transaction execution // with a given hash. It it used for import/export data as transactions are not persisted // on blockchain state after an upgrade. type TransactionLogsstruct { Hash ethcmn. Hashjson:"hash" Logs[ ] * ethtypes. Logjson:"logs" } Chain Config

TheChainConfig is a custom type that contains the same fields as the go-ethereumChainConfig parameters, but usingsdk.Int types instead of*big.Int . It also defines additional YAML tags for pretty printing.

TheChainConfig type is not a configurable SDK Param since the SDK does not allow for validation against a previous stored parameter values or Context fields. Since most of this type's fields rely on the block height value, this limitation prevents the validation of of potential new parameter values against the current block height (eg: to prevent updating the config block values to a past block).

If you want to update the config values, use an software upgrade procedure.

// ChainConfig defines the Ethereum ChainConfig parameters using sdk.Int values instead of big.Int. // // NOTE 1: Since empty/uninitialized Ints (i.e with a nil big.Int value) are parsed to zero, we need to manually // specify that negative Int values will be considered as nil. See getBlockValue for reference. // // NOTE 2: This type is not a configurable Param since the SDK does not allow for validation against // a previous stored parameter values or the current block height (retrieved from context). If you // want to update the config values, use an software upgrade procedure. type ChainConfigstruct { HomesteadBlock sdk. Intjson:"homestead_block" yaml:"homestead_block" // Homestead switch block (< 0 no fork, 0 = already homestead) DAOForkBlock sdk. Intjson:"dao_fork_block" yaml:"dao_fork_block" // TheDAO hard-fork switch block (< 0 no fork) DAOForkSupportbool json:"dao_fork_support" yaml:"dao_fork_support" // Whether the nodes supports or opposes the DAO hard-fork // EIP150 implements the Gas price changes (https://github.com/ethereum/EIPs/issues/150) EIP150Block sdk. Intjson:"eip150_block" yaml:"eip150_block" // EIP150 HF block (< 0 no fork) EIP150Hashstringjson:"eip150_hash" yaml:"eip150_hash" // EIP150 HF hash (needed for header only clients as only gas pricing changed) EIP155Block sdk. Intjson:"eip155_block" yaml:"eip155_block" // EIP155 HF block EIP158Block sdk. Intjson:"eip158_block" yaml:"eip158_block" // EIP158 HF block ByzantiumBlock sdk. Intjson:"byzantium_block" yaml:"byzantium_block" // Byzantium switch block (< 0 no fork, 0 = already on byzantium) ConstantinopleBlock sdk. Intjson:"constantinople_block" yaml:"constantinople_block" // Constantinople switch block (< 0 no fork, 0 = already activated) PetersburgBlock sdk. Intjson:"petersburg_block" yaml:"petersburg_block" // Petersburg switch block (< 0 same as Constantinople) IstanbulBlock sdk. Intjson:"istanbul_block" yaml:"istanbul_block" // Istanbul switch block (< 0 no fork, 0 = already on istanbul) MuirGlacierBlock sdk. Intjson:"muir_glacier_block" yaml:"muir_glacier_block" // Eip-2384 (bomb delay) switch block (< 0 no fork, 0 = already activated) YoloV2Block sdk. Intjson:"yoloV2_block" yaml:"yoloV2_block" // YOLO v1: https://github.com/ethereum/EIPs/pull/2657 (Ephemeral testnet) EWASMBlock sdk. Intjson:"ewasm_block" yaml:"ewasm_block" // EWASM switch block (< 0 no fork, 0 = already activated) }

## #

Config

EVM specific app.toml configuration.

### EVM Configuration

[ evm ]

# Tracer defines the 'vm.Tracer' type that the EVM will use when the node is run in

# debug mode. To enable tracing use the '--evm.tracer' flag when starting your node.

# Valid types are: json|struct|access_list|markdown

# tracer

""

### JSON RPC Configuration

[ json-rpc ]

# Enable defines if the gRPC server should be enabled.

# enable

true

# Address defines the EVM RPC HTTP server address to bind to.

# address

"0.0.0.0:8545"

# Address defines the EVM WebSocket server address to bind to.

# ws-address

"0.0.0.0:8546"

# API defines a list of JSON-RPC namespaces that should be enabled

# Example: "eth,txpool,personal,net,debug,web3"

# api

"eth,net,web3"

# GasCap sets a cap on gas that can be used in eth_call/estimateGas (0=infinite). Default: 25,000,000.

# gas-cap

25000000

# EVMTimeout is the global timeout for eth_call. Default: 5s.

# evm-timeout

"5s"

# TxFeeCap is the global tx-fee cap for send transaction. Default: 1eth.

## txfee-cap

1

## FilterCap sets the global cap for total number of filters that can be created

## filter-cap

200

## FeeHistoryCap sets the global cap for total number of blocks that can be fetched

## feehistory-cap

100

## LogsCap defines the max number of results can be returned from single 'eth_getLogs' query.

## logs-cap

10000

## BlockRangeCap defines the max block range allowed for 'eth_getLogs' query.

## block-range-cap

10000