paritytech / parity

Chain specification

Afri edited this page 26 days ago · 24 revisions

By default, when simply running `parity`, Parity Ethereum will connect to the official public Ethereum network.

In order to run a chain different to the official public Ethereum one, Parity has to be ran with the `--chain` option or with a config file specifying `chain = "path"` under `[parity]`. There are a few named presets that can be selected from or a custom JSON spec file can be supplied.

Chain presets available

- `mainnet` (default) main Ethereum network
- `kovan | testnet` the fast Ethereum test network
- `ropsten` the old Ethereum test network
- `classic` Ethereum Classic network
- `classic-testnet` original Morden testnet and current Ethereum Classic testnet
- `expanse` Expanse network
- `dev` a Private development chain to be used locally, submitted transactions are inserted into blocks instantly without the need to mine

Private chains

Parity can be used to set up a private chain. In addition to the usual Proof of Work Chains, Parity also includes Proof of Authority Chains which do not require mining. More details on the available options can be found on the Pluggable Consensus page.

JSON chain spec format

A JSON file which specifies rules of a blockchain, some fields are optional which are described following the minimal example, these default to 0.

```
{
	"name": "CHAIN_NAME",
	"engine": {
		"ENGINE_NAME": {
			"params": {
				ENGINE_PARAMETERS
			}
		}
	},
	"genesis": {
		"seal": {
			ENGINE_SPECIFIC_GENESIS_SEAL
		},
		"difficulty": "0x20000",
		"gasLimit": "0x2fefd8"
	},
	"params": {
			"networkID" : "0x2",
			"maximumExtraDataSize": "0x20",
			"minGasLimit": "0x1388"
	},
	"accounts": {
		GENESIS_ACCOUNTS
	}
}
```

- `"name"` field contains any name used to identify the chain.

- `"engine"` field describes the consensus engine used for a particular chain, details are explained in the Consensus Engines section.
- `"genesis"` contains the genesis block (first block in the chain) header information.
  - `"seal"` is consensus engine specific and is further described in Consensus Engines.
  - `"difficulty"` difficulty of the genesis block, matters only for PoW chains.
  - `"gasLimit"` gas limit of the genesis, affects the initial gas limit adjustment.

  Optional:

  - `"author"` address of the genesis block author.
  - `"timestamp"` UNIX timestamp of the genesis block.
  - `"parentHash"` hash of the genesis "parent" block.
  - `"transactionsRoot"` genesis transactions root.
  - `"receiptsRoot"` genesis receipts root.
  - `"stateRoot"` genesis state root, calculated automatically from the `"accounts"` field.
  - `"gasUsed"` gas used in the genesis block.
  - `"extraData"` extra data of the genesis block.
- `"params"` contains general chain parameters:
  - `"networkID"` DevP2P supports multiple networks, and ID is used to uniquely identify each one which is used to connect to correct peers and to prevent transaction replay across chains.
  - `"maximumExtraDataSize"` determines how much extra data the block issuer can place in the block header.
  - `"minGasLimit"` gas limit can adjust across blocks, this parameter determines the absolute minimum it can reach.

  Optional:

  - `"accountStartNonce"` in the past this was used for transaction replay protection
  - `"chainID"` chain identifier, if not present then equal to `networkID`
  - `"subprotocolName"` by default its the `eth` subprotocol
  - `"forkBlock"` block number of the latest fork that should be checked
  - `"forkCanonHash"` hash of the canonical block at `forkBlock`
  - `"bombDefuseTransition"` block number at which the difficulty bomb (epsilon in Yellow Paper Eqs. 39, 44) is removed from the difficulty evolution
- `"accounts"` contains optional contents of the genesis block, such as simple accounts with balances or contracts. Parity does not include the standard Ethereum builtin contracts by default. These are necessary when writing new contracts in Solidity, since compiled Solidity often refers to them. To make the chain behave like the public Ethereum chain the 4 contracts need to be included in the spec file, as shown in the example below:

```
"accounts": {
    "0x0000000000000000000000000000000000000001": { "balance": "1",
"builtin": { "name":   "ecrecover", "pricing": { "linear": { "base": 3000,
"word": 0 } } } },
    "0x0000000000000000000000000000000000000002": { "balance": "1",
"builtin": { "name": "sha256", "pricing": { "linear": { "base": 60, "word":
12 } } } },
    "0x0000000000000000000000000000000000000003": { "balance": "1",
"builtin": { "name": "ripemd160", "pricing": { "linear": { "base": 600,
"word": 120 } } } },
    "0x0000000000000000000000000000000000000004": { "balance": "1",
"builtin": { "name": "identity", "pricing": { "linear": { "base": 15, "word":
3 } } } }
}
```


Clone this wiki locally

https://github.com/parity

Clone in Desktop

Other types of accounts that can be specified:

- simple accounts with some balance `"0x...": { "balance": "100000000000" }`
- full account state `"0x...": { "balance": "100000000000", "nonce": "0", "code": "0x...", "storage": { "0": "0x...", ... } }`
- contract constructor, similar to sending a transaction with bytecode `"0x...": { "balance": "100000000000", "constructor": "0x..." }`

Optional spec fields:

- **`"dataDir"`** sets a name of the chain to be used in the data directory instead of the chain name
- **`"nodes"`** a list of boot nodes in the enode format

Coming from Geth

To connect to a Geth node or just use the same network setup you can use the genesis converter to generate a Parity compatible chain specification file or the preconfigured cross-client chain specifications to get started quickly. The chain spec can be then used by supplying it to the `--chain` Parity option.

To replicate some of the more obscure bugs from Geth's RPC, `--geth` option can be used; be warned, this disables some of the more advanced Parity features so only use it if you know you have to.

FAQ | Wiki Index | Stack Exchange [+] | Github Repository [+] | Gitter Chat [+] | Parity Website [+]