Cheatcalls EIP

Standardizing JSON-RPC interface for development nodes

Kris Kaczor

Cofounder PhoenixLabs, Contributor SparkDAO

Ethereum nodes landscape

Production nodes

geth, nethermind, erigon, reth, prysm…

Development nodes

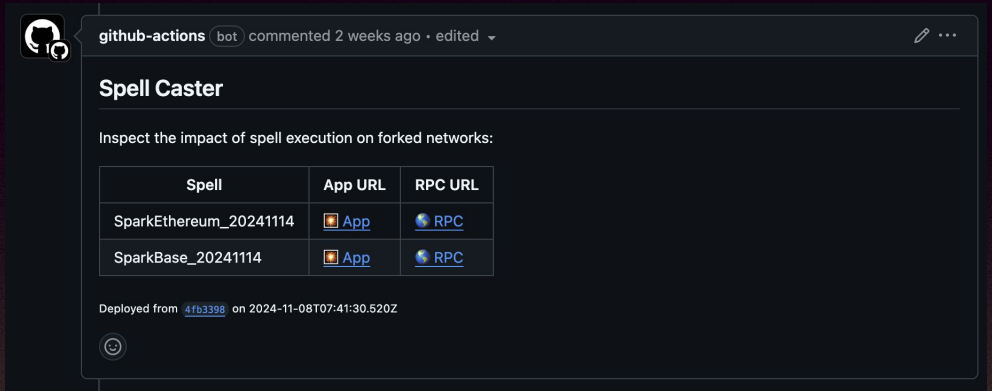
anvil (foundry), hardhat node, tenderly, build bear

Test environments

foundry, hardhat

Why development nodes are so useful?

* Can be local or running in the cloud (devnet)
* Use cases:
  * Forking support
  * **Simulating governance spell execution**
  * E2E / **Integration Tests** for different systems
  * Private testnets
* Limited control with **special** JSON-RPC methods



Special JSON-RPC calls

* evm_setNextBlockTimestamp
* evm_revert
* tenderly_setErc20Balance
* anvil_mine
* . . .

Cheatcalls EIP

Cheatcalls EIP

* **Standard to be implemented** by Development nodes
* Like Foundry’s cheatcodes but for json-rpc
* Increases nodes interoperability
* Test suite to verify spec conformance
* Share **cheat_** prefix

* Gives full control over the node to developers
  * Time management
  * Mining control
  * Snapshots
  * Impersonation
  * Balance and storage management

Time management

* **cheat_increaseTime(deltaSeconds: Quantity)**
  * Mines a new block with a timestamp of lastTimestamp + deltaSeconds
* **cheat_setNextBlockTimestamp (nextTimestamp: Quantity |’default’)**
  * Does not mine a new block, but once new block is mined, it will have timestamp of exactly nextTimestamp. Any methods reading state such as eth_call respects new timestamp when queried for 'pending' block.
  * To unset, call with ‘default’

Mining control

* cheat_mine(count: Quantity = 1, gapSeconds: Quantity = 1)
* cheat_mining_mode(mode: Mode)
* cheat_dropTransaction(hash: Data)

```
type InputMiningMode =
  | { type: "auto" }
  | { type: "manual"; ordering?: MiningOrdering }
  | { type: "interval"; intervalSeconds: Quantity;
ordering?: MiningOrdering };

type MiningOrdering =
  | "highest-fee-first" // default
  | "oldest-first"
  | "random";
```

Snapshots

* **cheat_snapshot(): Data**
  * Snapshots current state of the blockchain, including Cheatcalls related state like timestamp of a next branch. Returned value can be any hex string but has to be unique.
* **cheat_revertSnapshot(id: Data)**
  * Reverts to a given snapshot. Throws if snapshot id was not found. Revert multiple times to the same snapshot MUST be supported.

Impersonation

* `cheat_impersonateAllAccounts()`
* `cheat_stopImpersonatingAllAccounts()`
* `eth_sendTransaction` vs `eth_sendRawTransaction`

Meta

* cheat_info()
  * Returns status of the whole node and cheatcodes (next timestamp, mining mode etc.)

```
interface CheatcallsInfo {
  cheatcallsSpecVersion: string;
  runMode: RunMode;
  miningMode: MiningMode;
  impersonateAllEnabled: boolean;
  nextBlockTimestamp: Quantity;
  minGasPrice: Quantity;
  gasLimit: Quantity;
  nextBlockBaseFeePerGas: Quantity;
}
```

Balance and storage management

* cheat_setBalance(account: Address, balanceInWei: Quantity)
* cheat_setErc20Balance(token: Address, account: Address, balanceInBaseUnit: Quantity)
  * Best effort implementation
* cheat_setStorageAt(account: Address,slot: Data, value: Quantity)

Tangent: How does setting ERC20 balance work?

* Problem: we can tweak arbitrary storage slots but we don’t know which slot holds particular user ERC20 balance
* EVM Storage is a huge map **uint256 -> uint256** scoped for each contract
* Solidity storage layout
  * Each property has an order dependent index
  * Mappings select slots based on a hash of an index and keys

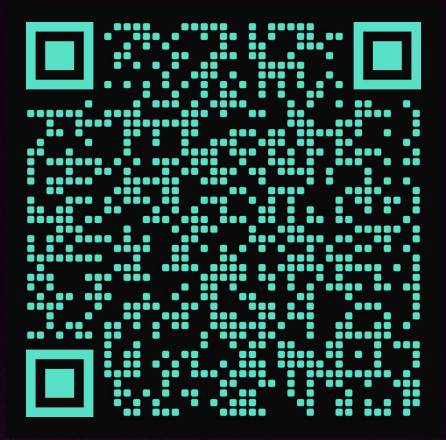
Tangent: How does setting ERC20 balance work? (part 2)

1. Trace **balanceOf(account)** read - **eth_createAccessList**
2. Find all accessed storage slots
3. Tweak **{i}** accessed storage location with random value {x}
4. execute **balanceOf(account)** again,
5. If **{x}** was returned we **{i}** is the storage slot that we were looking for
6. If not go back to step 3 and check next storage slot
7. Set final value on slot {i}

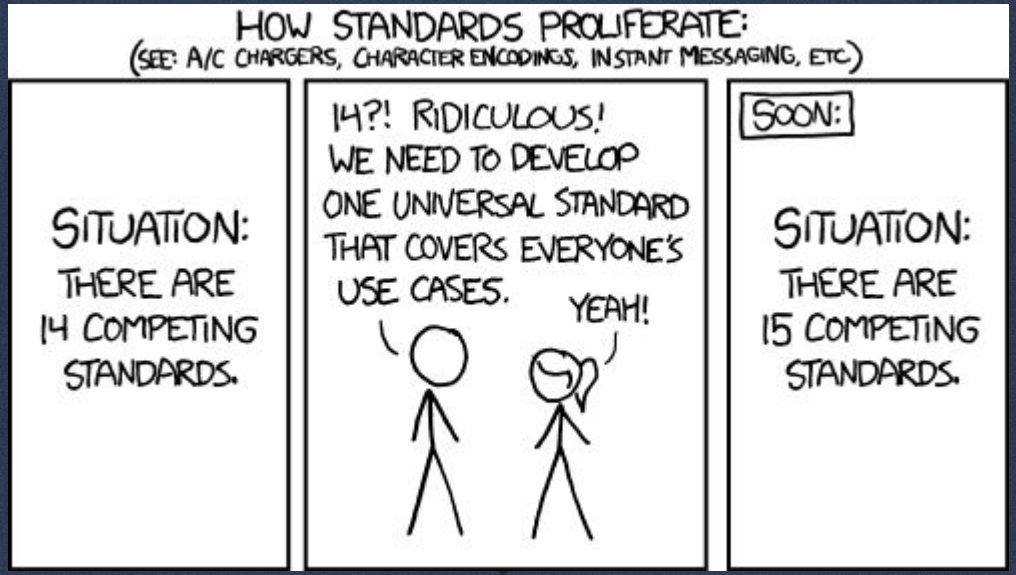
Example implementation: [forge-std/StdStorage]

Current state of the EIP

* Pre-draft ready and submitted to eth-magicians
* 3 contributors and more reviewers
  * Special thanks to: **Piotr Szlachciak** and **Emmanuel Antony**
  * Foundry, Hardhat, Tenderly, BuildBear teams are aware



Future of the EIP



Alternative, client side approach

Viem / ethers.js client

- Best effort attempt to reduce incompatibilities
- Many features won't be available
- Can warn users about incompatibilities
- Use viem-deal like approach to implement **setErc20Balance** user side

Thank you!

Kris Kaczor

Cofounder PhoenixLabs, Contributor SparkDAO

@krzkaczor