Cheat Codes

Introduction

To help with testing, the sandbox is shipped with a set of cheatcodes.

Cheatcodes allow you to change the time of the Aztec block, load certain state or more easily manipulate Ethereum instead of having to write dedicated RPC calls to anvil or hardhat.

Prerequisites If you aren't familiar withAnvil , we recommend reading up on that since Aztec Sandbox uses Anvil as the local Ethereum instance.

Aims

The guide will cover how to manipulate the state of the:

- Ethereum blockchain;
- Aztec network.

Dependencies

For this guide, the following Aztec packages are used:

- @aztec/aztec.js

Initialization

import

{ createPXEClient , CheatCodes }

from

"@aztec/aztec.js" ; const pxeRpcUrl =

"http://localhost:8080" ; const ethRpcUrl =

"http://localhost:8545" ; const pxe =

createPXEClient (pxeRpcUrl) ; const cc =

await CheatCodes . create (ethRpcUrl , pxe) ; There are two properties of the CheatCodes class -eth andaztec for cheatcodes relating to the Ethereum blockchain (L1) and the Aztec network (L2) respectively.

Ethereum related cheatcodes

These are cheatcodes exposed from anvil/hardhat conveniently wrapped for ease of use in the Sandbox.

Interface

// Fetch current block number of Ethereum public

async

blockNumber ( ) :

Promise < number

// Fetch chain ID of the local Ethereum instance public

async

chainId ( ) :

Promise < number

// Fetch current timestamp on Ethereum public

async

timestamp ( ) :

Promise < number

// Mine a given number of blocks on Ethereum. Mines 1 block by default public

async

mine (numberOfBlocks =

1) :

Promise < void

// Set the timestamp for the next block on Ethereum. public

async

setNextBlockTimestamp (timestamp :

number) :

Promise < void

// Dumps the current Ethereum chain state to a given file. public

async

dumpChainState (fileName :

string) :

Promise < void

// Loads the Ethereum chain state from a file. You may use`dumpChainState()` to save the state of the Ethereum chain to a file and later load it. public

async

loadChainState (fileName :

string) :

Promise < void

// Load the value at a storage slot of a contract address on Ethereum public

async

load (contract : EthAddress , slot : bigint) :

Promise < bigint

// Set the value at a storage slot of a contract address on Ethereum (e.g. modify a storage variable on your portal contract or even the rollup contract). public

async

store (contract : EthAddress , slot : bigint , value : bigint) :

Promise < void

// Computes the slot value for a given map and key on Ethereum. A convenient wrapper to find the appropriate storage slot to load or overwrite the state. public

keccak256 (baseSlot : bigint , key : bigint) : bigint

// Let you send transactions on Ethereum impersonating an externally owned or contract, without knowing the private key. public

async

startImpersonating (who : EthAddress) :

Promise < void

// Stop impersonating an account on Ethereum that you are currently impersonating. public

async

stopImpersonating (who : EthAddress) :

Promise < void

// Set the bytecode for a Ethereum contract public

async

etch (contract : EthAddress , bytecode :

0x { string }) :

Promise < void

// Get the bytecode for a Ethereum contract public

async

getBytecode (contract : EthAddress) :

Promise < 0x { string }

blockNumber

Function Signature

public

async

blockNumber ( ) :

Promise < number

Description

Fetches the current Ethereum block number.

Example

const blockNumber =

await cc . eth . blockNumber ( ) ;

chainId

Function Signature

public

async

chainId ( ) :

Promise < number

Description

Fetches the Ethereum chain ID

Example

const chainId =

await cc . eth . chainId ( ) ;

timestamp

Function Signature

public

async

timestamp ( ) :

Promise < number

Description

Fetches the current Ethereum timestamp.

Example

const timestamp =

await cc . eth . timestamp ( ) ;

mine

Function Signature

public

async

mine (numberOfBlocks =

1) :

Promise < void

Description

Mines the specified number of blocks on Ethereum (default 1).

Example

const blockNum =

await cc . eth . blockNumber ( ) ; await cc . eth . mine (10) ;

// mines 10 blocks const newBlockNum =

await cc . eth . blockNumber ( ) ;

// = blockNum + 10.

setNextBlockTimestamp

Function Signature

public

async

setNextBlockTimestamp (timestamp :

number) :

Promise < void

Description

Sets the timestamp (unix format in seconds) for the next mined block on Ethereum. Time can only be set in the future. If you set the timestamp to a time in the past, this method will throw an error.

Example

// // Set next block timestamp to 16 Aug 2023 10:54:30 GMT await cc . eth . setNextBlockTimestamp (1692183270) ; // next transaction you will do will have the timestamp as 1692183270

dumpChainState

Function Signature

public

async

dumpChainState (fileName :

string) :

Promise < void

Description

Dumps the current Ethereum chain state to a file. Stores a hex string representing the complete state of the chain in a file with the provided path. Can be re-imported into a fresh/restarted instance of Anvil to reattain the same state. When combined withloadChainState() cheatcode, it can be let you easily import the current state of mainnet into the Anvil instance of the sandbox.

Example

await cc . eth . dumpChainState ("chain-state.json") ;

loadChainState

Function Signature

public

async

loadChainState (fileName :

string) :

Promise < void

Description

Loads the Ethereum chain state from a file which contains a hex string representing an Ethereum state. When given a file previously written to bycc.eth.dumpChainState() , it merges the contents into the current chain state. Will overwrite any colliding accounts/storage slots.

Example

await cc . eth . loadChainState ("chain-state.json") ;

load

Function Signature

public

async

load (contract : EthAddress , slot : bigint) :

Promise < bigint

Description

Loads the value at a storage slot of a Ethereum contract.

Example

contract

LeetContract

{ uint256

private leet =

1337 ;

// slot 0 } const leetContractAddress = EthAddress . fromString ("0x1234...") ; const value =

await cc . eth . load (leetContractAddress ,

BigInt (0)) ; console . log (value) ;

// 1337

store

Function Signature

public

async

store (contract : EthAddress , slot : bigint , value : bigint) :

Promise < void

Description

Stores the value in storage slot on a Ethereum contract.

Example

contract

LeetContract

{ uint256

private leet =

1337 ;

// slot 0 } const leetContractAddress = EthAddress . fromString ("0x1234...") ; await cc . eth . store (leetContractAddress ,

BigInt (0) ,

BigInt (1000)) ; const value =

await cc . eth . load (leetContractAddress ,

BigInt (0)) ; console . log (value) ;

// 1000

keccak256

Function Signature

public

keccak256 (baseSlot : bigint , key : bigint) : bigint

Description

Computes the storage slot for a map key.

Example

contract

LeetContract

{ uint256

private leet =

1337 ;

// slot 0 mapping (address

=>

uint256)

public balances ;

// base slot 1 } // find the storage slot for key0xdead in the balance map. const address =

BigInt ("0x000000000000000000000000000000000000dead") ; const slot = cc . eth . keccak256 (1n , address) ; // store balance of 0xdead as 100 await cc . eth . store (contractAddress , slot ,

100n) ;

startImpersonating

Function Signature

public

async

startImpersonating (who : EthAddress) :

Promise < void

Description

Start impersonating an Ethereum account. This allows you to use this address as a sender.

Example

await cc . eth . startImpersonating (EthAddress . fromString (address)) ;

stopImpersonating

Function Signature

public

async

stopImpersonating (who : EthAddress) :

Promise < void

Description

Stop impersonating an Ethereum account. Stops an active impersonation started by startImpersonating.

Example

await cc . eth . stopImpersonating (EthAddress . fromString (address)) ;

getBytecode

Function Signature

public

async

getBytecode (contract : EthAddress) :

Promise < 0x { string }

Description

Get the bytecode for an Ethereum contract.

Example

const bytecode =

await cc . eth . getBytecode (contract) ;

// 0x6080604052348015610010...

etch

Function Signature

public

async

etch (contract : EthAddress , bytecode :

0x { string }) :

Promise < void

Description

Set the bytecode for an Ethereum contract.

Example

const bytecode =

0x6080604052348015610010... ; await cc . eth . etch (contract , bytecode) ; console . log (await cc . eth . getBytecode (contract)) ;

// 0x6080604052348015610010...

Aztec related cheatcodes

These are cheatcodes specific to manipulating the state of Aztec rollup.

Interface

// Get the current aztec block number public

async

blockNumber ( ) :

Promise < number

// Set time of the next execution on aztec. It also modifies time on Ethereum for next execution and stores this time as the last rollup block on the rollup contract. public

async

warp (to :

number) :

Promise < void

// Loads the value stored at the given slot in the public storage of the given contract. public

async

loadPublic (who : AztecAddress , slot : Fr | bigint) :

Promise < Fr

// Loads the value stored at the given slot in the private storage of the given contract. public

async

loadPrivate (owner : AztecAddress , contract : AztecAddress , slot : Fr | bigint) :

Promise < Note [ ]

// Computes the slot value for a given map and key. public

computeSlotInMap (baseSlot : Fr | bigint , key : Fr | bigint) : Fr

blockNumber

Function Signature

public

async

blockNumber ( ) :

Promise < number

Description

Get the current aztec block number.

Example

const blockNumber =

await cc . aztec . blockNumber ( ) ;

warp

Function Signature

public

async

warp (to :

number) :

Promise < void

Description

Sets the time on Ethereum and the time of the next block on Aztec. Like with the corresponding Ethereum cheatcode, time can only be set in the future, not the past. Otherwise, it will throw an error.

Example

const timestamp =

await cc . eth . timestamp ( ) ; const newTimestamp = timestamp +

100_000_000 ; await cc . aztec . warp (newTimestamp) ; // any Aztec.nr contract calls that make use of current timestamp
// and is executed in the next rollup block will now read newTimestamp

computeSlotInMap

Function Signature

public

computeSlotInMap (baseSlot : Fr | bigint , key : Fr | bigint) : Fr

Description

Compute storage slot for a map key. The baseSlot is specified in the Aztec.nr contract.

Example

struct

Storage

{ balances :

Map < AztecAddress ,

PublicMutable < Field

, }

impl

Storage

{ fn

init ( )

->

Self

{ Storage

{ balances :

Map :: new (1 ,

| slot |

PublicMutable :: new (slot)) , } } }

contract Token

{ ... } const slot = cc . aztec . computeSlotInMap (1n , key) ;

loadPublic

Function Signature

public

async

loadPublic (who : AztecAddress , slot : Fr | bigint) :

Promise < Fr

Description

Loads the value stored at the given slot in the public storage of the given contract.

Note: One Field element occupies a storage slot. Hence, structs with multiple field elements will be spread over multiple sequential slots. Using loadPublic will only load a single field of the struct (depending on the size of the attributes within it).

Example

struct

Storage

{ balances :

Map < AztecAddress ,

PublicMutable < Field

, }

impl

Storage

{ fn

init (context :

Context)

->

Self

{ Storage

{ balances :

Map :: new (context ,

1 ,

| context , slot |

PublicMutable :: new (context , slot)) , } } }

contract Token

{ ... } const address = AztecAddress . fromString ("0x123...") ; const slot = cc . aztec . computeSlotInMap (1n , key) ; const value =

await cc . aztec . loadPublic (address , slot) ;

loadPrivate

Function Signature

public

async

loadPrivate (owner : AztecAddress , contract : AztecAddress , slot : Fr | bigint) :

Promise < Note [ ]

Description

Loads the value stored at the given slot in the private storage of the given contract.

Note: One Field element occupies a storage slot. Hence, structs with multiple field elements will be spread over multiple sequential slots. Using loadPublic will only load a single field of the struct (depending on the size of the attributes within it).

Example

struct

Storage

{ ... pending_shields :

Set < TransparentNote ,

TRANSPARENT_NOTE_LEN

, }

impl

Storage

{ fn

init ( )

->

Self

{ Storage

{ ... pending_shields :

Set :: new (context ,

5 ,

TransparentNoteMethods) , } } }

contract Token

{ ... } load_private_cheatcode const mintAmount =

100n ; const secret = Fr . random ( ) ; const secretHash =

computeMessageSecretHash (secret) ; const receipt =

await token . methods . mint_private (mintAmount , secretHash) . send ( ) . wait ( ) ;

const note =

new

Note ([new

Fr (mintAmount) , secretHash]) ; const pendingShieldStorageSlot =

new

Fr (5n) ; const noteTypeId =

new

Fr (84114971101151129711410111011678111116101n) ;

// TransparentNote const extendedNote =

new

ExtendedNote (note , admin . address , token . address , pendingShieldStorageSlot , noteTypeId , receipt . txHash ,) ;
await pxe . addNote (extendedNote) ;

// check if note was added to pending shield: const notes =

await cc . aztec . loadPrivate (admin . address , token . address , pendingShieldStorageSlot) ; const values = notes . map (note => note . items [0]) ; const balance = values . reduce ((sum , current)

=> sum + current . toBigInt ( ) ,

0n) ; expect (balance) . toEqual (mintAmount) Source code: yarn-project/end-to-end/src/e2e_cheat_codes.test.ts#L211-L237

Participate

Keep up with the latest discussion and join the conversation in the Aztec forum .

You can also use the above link to request more cheatcodes.Edit this page

Previous Sandbox Reference Next Private Execution Environment (PXE)