Environment

Every method execution has an environment associated with information such as:

1. Who called the method
2. How much money is attached to the call
3. How many computational resources are available
4. The current timestamp
5. Helper functions for Public Key derivation, for example

Environment Variables

- JavaScript
- Rust

Variable Name SDK Variable Description Predecessor near.predecessorAccountId() Account ID that called this method Current Account near.currentAccountId() Account ID of this smart contract Signer near.signerAccountId() Account ID that signed the transaction leading to this execution Attached Deposit near.attachedDeposit() Amount in NEAR attached to the call by the predecessor Account Balance near.accountBalance() Balance of this smart contract (including Attached Deposit) Prepaid Gas near.prepaidGas() Amount of gas available for execution Timestamp near.blockTimestamp() Current timestamp (number of non-leap-nanoseconds since January 1, 1970 0:00:00 UTC) Current Epoch near.epochHeight() Current epoch in the blockchain Block Index near.blockIndex() Current block index (a.k.a. block height) Storage Used near.storageUsage() Current storage used by this smart contract Used Gas near.usedGas() Amount of gas used for execution Signer Public Key near.signerAccountPk() Sender Public Key Account Locked Balance near.accountLockedBalance() Balance of this smart contract that is locked Variable Name SDK Variable Description Predecessor env::predecessor_account_id() Account ID that called this method Current Account env::current_account_id() Account ID of this smart contract Signer env::signer_account_id() Account ID that signed the transaction leading to this execution Attached Deposit env::attached_deposit() Amount in NEAR attached to the call by the predecessor Account Balance env::account_balance() Balance of this smart contract (including Attached Deposit) Prepaid Gas env::prepaid_gas() Amount of gas available for execution Timestamp env::block_timestamp() Current timestamp (number of non-leap-nanoseconds since January 1, 1970 0:00:00 UTC) Current Epoch env::epoch_height() Current epoch in the blockchain Block Index env::block_index() Current block index (a.k.a. block height) Storage Used env::storage_usage() Current storage used by this smart contract in bytes Storage Byte Cost env::storage_byte_cost() Current storage cost per byte in yoctoNEAR Used Gas env::used_gas() Amount of gas used for execution Signer Public Key env::signer_account_pk() Sender Public Key Account Locked Balance env::account_locked_balance() Balance of this smart contract that is locked

Who is Calling? Who am I?

The environment gives you access to 3 important users: thecurrent_account , thepredecessor , and thesigner .

Current Account

Thecurrent_account contains the address in which your contract is deployed. This is very useful to implement ownership, e.g. making a public method only callable by the contract itself.

Predecessor and Signer

Thepredecessor is the account that called the method in the contract. Meanwhile, thesigner is the account thatsigned the initial transaction.

During a simple transaction (nocross-contract calls) thepredecessor is the same as thesigner . For example, ifalice.near callscontract.near , from the contract's perspective,alice.near is both thesigner and thepredecessor . However, ifcontract.near creates across-contract call , then thepredecessor changes down the line. In the example below, whenpool.near executes, it would seecontract.near as thepredecessor andalice.near as thesigner .

You can access information about the users interacting with your smart contract

tip In most scenarios you willonly need to know the predecessor . However, there are situations in which the signer is very useful. For example, when addingNFTs intothis marketplace , the contract checks that thesigner , i.e. the person who generated the transaction chain, is the NFT owner.

Balances and Attached NEAR

The environment gives you access to 3 token-related parameters, all expressed in yoctoNEAR (1 Ⓝ = 1024 yⓃ):

Attached Deposit

attached_deposit represents the amount of yoctoNEAR the predecessor attached to the call.

This amount isalready deposited in your contract's account, and isautomatically returned to thepredecessor if yourmethod panics .

warning If you make across-contract call and it panics, the funds are sent back toyour contract . See how to handle this situation in thecallback section

Account Balance

account_balance represents the balance of your contract (current_account).

It includes theattached_deposit , since it was deposited when the method execution started.

If the contract has any locked NEAR, it will appear inaccount_locked_balance .

Storage Used

storage_used represents the amount ofstorage that is currently being used by your contract.

tip If you want to know how much storage a structure uses, print the storage before and after storing it.

Telling the Time

The environment exposes three different ways to tell the pass of time, each representing a different dimension of the underlying blockchain.

Timestamp

Thetimestamp attribute represents the approximatedUNIX timestamp at which this call was executed. It quantifies time passing in a human way, enabling to check if a specific date has passed or not.

Current Epoch

The NEAR blockchain groups blocks inEpochs . Thecurrent_epoch attribute measures how many epochs have passed so far. It is very useful to coordinate with other contracts that measure time in epochs, such as thevalidators .

Block Index

Theblock_index represents the index of the block in which this transaction will be added to the blockchain.

Gas

Your contract has alimited number of computational resources to use on each call. Such resources are measured inGas .

Gas can be thought of as wall time, where 1 PetaGas (1_000 TGas) is ~1 second of compute time.

Each code instruction costs a certain amount of Gas, and if you run out of it, the execution halts with the error messageExceeded the prepaid gas .

The environment gives you access to two gas-related arguments:prepaid_gas andused_gas .

Prepaid Gas

prepaid_gas represents the amount of Gas thepredecessor attached to this call. It cannot exceed the limit 300TGas (300 * 1012 Gas).

Used Gas

used_gas contains the amount of Gas that has been used so far. It is useful to estimate the Gas cost of running a method.

warning Duringcross-contract calls always make sure the callback has enough Gas to fully execute. tip If you alreadyestimated the Gas a method needs, you can ensure it never runs out of Gas by usingassert

- Rust

const

REQUIRED_GAS :

Gas

=

Gas (20_000_000_000_000) ;

// 20 TGas assert! (env :: prepaid_gas ( )

=

REQUIRED_GAS ,

"Please attach at least 20 TGas") ;

Environment Functions

- JavaScript
- Rust

Function Name SDK method Description SHA 256 near.sha256(value) Hashes a sequence of bytes using sha256. Keccak 256 near.keccak256(value) Hashes a sequence of bytes using keccak256. Keccak 512 near.keccak512(value) Hashes a sequence of bytes using keccak512. RIPEMD 160 near.ripemd160(value) Hashes the bytes using the RIPEMD-160 hash function. EC Recover near.ecrecover(hash, sig, v, malleabilityFlag) Recovers an ECDSA signer address from a 32-byte messagehash and a correspondingsignature along withv recovery byte. Takes in an additional flag to check for malleability of the signature which is generally only ideal for transactions. Returns 64 bytes representing the public key if the recovery was successful. Log String near.log(msg) Logs the string message. This message is stored on chain. Validator Stake near.validatorStake(accountId) For a given account return its current stake. If the account is not a validator, returns 0. Validator Total Stake near.validatorTotalStake() Returns the total stake of validators in the current epoch. Function Name SDK method Description SHA 256 env::sha256(value) Hashes a sequence of bytes using sha256. Keccak 256 env::keccak256(value) Hashes a sequence of bytes using keccak256. Keccak 512 env::keccak512(value) Hashes a sequence of bytes using keccak512. SHA 256 (Array) env::sha256_array(value) Hashes the bytes using the SHA-256 hash function. This returns a 32 byte hash. Keccak 256 (Array) env::keccak256_array(value) Hashes the bytes using the Keccak-256 hash function. This returns a 32 byte hash. Keccak 512 (Array) env::keccak512_array(value) Hashes the bytes using the Keccak-512 hash function. This returns a 64 byte hash. RIPEMD 160 (Array) env::ripemd160_array(value) Hashes the bytes using the RIPEMD-160 hash function. This returns a 20 byte hash. EC Recover env::ecrecover(hash, signature, v, malleability_flag) Recovers an ECDSA signer address from a 32-byte messagehash and a correspondingsignature along withv recovery byte. Takes in an additional flag to check for malleability of the signature which is generally only ideal for transactions. Returns 64 bytes representing the public key if the recovery was successful. Panic String env::panic_str(message) Terminates the execution of the program with the UTF-8 encoded message. Log String env::log_str(message) Logs the string message. This message is stored on chain. Validator Stake env::validator_stake(account_id) For a given account return its current stake. If the account is not a validator, returns 0. Validator Total Stake env::validator_total_stake() Returns the total stake of validators in the current epoch. info In the JS SDK,throw new Error("message") mimics the behavior of Rust'senv::panic_str("message") . Edit this page Last updatedonDec 9, 2023 bygagdiez Was this page helpful? Yes No

Previous The Contract Class Next State & Data Structures