

# 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	<code>near.predecessorAccountId()</code>	Account ID that called this method
Current Account	<code>near.currentAccountId()</code>	Account ID of this smart contract
Signer	<code>near.signerAccountId()</code>	Account ID that signed the transaction leading to this execution
Attached Deposit	<code>near.attachedDeposit()</code>	Amount in NEAR attached to the call by the predecessor
Account Balance	<code>near.accountBalance()</code>	Balance of this smart contract (including Attached Deposit)
Prepaid Gas	<code>near.prepaidGas()</code>	Amount of gas available for execution
Timestamp	<code>near.blockTimestamp()</code>	Current timestamp (number of non-leap-nanoseconds since January 1, 1970 0:00:00 UTC)
Current Epoch	<code>near.epochHeight()</code>	Current epoch in the blockchain
Block Index	<code>near.blockIndex()</code>	Current block index (a.k.a. block height)
Storage Used	<code>near.storageUsage()</code>	Current storage used by this smart contract
Used Gas	<code>near.usedGas()</code>	Amount of gas used for execution
Signer Public Key	<code>near.signerAccountPk()</code>	Sender Public Key
Account Locked Balance	<code>near.accountLockedBalance()</code>	Balance of this smart contract that is locked

  

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

During a simple transaction (no [cross-contract calls](#)) `thepredecessor` is the same as `thesigner`. For example, if `alice.near` calls `contract.near`, from the contract's perspective, `alice.near` is both `thesigner` and `thepredecessor`. However, if `contract.near` creates a [cross-contract call](#), then `thepredecessor` changes down the line. In the example below, when `pool.near` executes, it would see `contract.near` as `thepredecessor` and `alice.near` as `thesigner`.

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

tip In most scenarios you will only need to know the predecessor. However, there are situations in which the signer is very useful. For example, when adding [NFTs](#) into [this marketplace](#), the contract checks that the signer, 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  $\text{N}$  = 1024  $\text{yN}$ ):

### Attached Deposit

attached\_deposit represents the amount of yoctoNEAR the predecessor attached to the call.

This amount is already deposited in your contract's account, and is automatically returned to the predecessor if your method panics.

warning If you make a [cross-contract call](#) and it panics, the funds are sent back to your contract. See how to handle this situation in the [callback section](#)

## Account Balance

account\_balance represents the balance of your contract (current\_account).

It includes the attached\_deposit, since it was deposited when the method execution started.

If the contract has any locked NEAR, it will appear in account\_locked\_balance.

## Storage Used

storage\_used represents the amount of [Storage](#) 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

The timestamp attribute represents the approximated UNIX 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 in [Epochs](#). The current\_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 the [validators](#).

### Block Index

The block\_index represents the index of the block in which this transaction will be added to the blockchain.

## Gas

Your contract has a limited number of computational resources to use on each call. Such resources are measured in [Gas](#).

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 message Exceeded the prepaid gas.

The environment gives you access to two gas-related arguments: prepaid\_gas and used\_gas.

### Prepaid Gas

prepaid\_gas represents the amount of Gas the predecessor attached to this call. It cannot exceed the limit 300 TGas (300 \* 10<sup>12</sup> 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 During [cross-contract calls](#) always make sure the callback has enough Gas to fully execute. tip If you already [estimated the Gas](#) a method needs, you can ensure it never runs out of Gas by using assert

- 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](#)