Best practices

Enable overflow checks

It's usually helpful to panic on integer overflow. To enable it, add the following into yourCargo.toml file:

[profile.release] overflow-checks = true

Userequire!

early

Try to validate the input, context, state and access usingrequire! before taking any actions. The earlier you panic, the moregas you will save for the caller.

[near_bindgen]

impl

Contract

{ pub

fn

set_fee (& mut

self , new_fee :

Fee)

{ require! (env :: predecessor_account_id ( )

==

self . owner_id ,

"Owner's method") ; new_fee . assert_valid ( ) ; self . internal_set_fee (new_fee) ; } } Note : If you want debug information in the panic message or if you are using an SDK version before4.0.0-pre.2 , the Rustassert! macro can be used instead ofrequire! .

[near_bindgen]

impl

Contract

{ pub

fn

set_fee (& mut

self , new_fee :

Fee)

{ assert_eq! (env :: predecessor_account_id ( ) ,

self . owner_id ,

"Owner's method") ; new_fee . assert_valid ( ) ; self . internal_set_fee (new_fee) ; } }

Uselog!

Use logging for debugging and notifying user.

When you need a formatted message, you can use the following macro:

log! ("Transferred {} tokens from {} to {}" , amount , sender_id , receiver_id) ; It's equivalent to the following message:

env :: log_str (format! ("Transferred {} tokens from {} to {}" , amount , sender_id , receiver_id) . as_ref ( )) ;

ReturnPromise

If your method makes a cross-contract call, you probably want to return the newly createdPromise . This allows the caller (such as a near-cli or near-api-js call) to wait for the result of the promise instead of returning immediately. Additionally, if the promise fails for some reason, returning it will let the caller know about the failure, as well as enabling NEAR Explorer and other tools to mark the whole transaction chain as failing. This can prevent false-positives when the first or first few transactions in a chain succeed but a subsequent transaction fails.

E.g.

[near_bindgen]

impl

Contract

{ pub

fn

withdraw_100 (& mut

self , receiver_id :

AccountId)

->

Promise

{ Promise :: new (receiver_id) . transfer (100) } }

Reuse crates fromnear-sdk

near-sdk re-exports the following crates:

- borsh
- base64
- bs58
- serde
- serde_json

Most common crates includeborsh which is needed for internal STATE serialization andserde for external JSON serialization.

When marking structs withserde::Serialize you need to use#[serde(crate = "near_sdk::serde")] to point serde to the correct base crate.

/// Import borsh from near_sdk crate use

near_sdk :: borsh :: { self ,

BorshDeserialize ,

BorshSerialize } ; /// Importserde from near_sdk crate use

near_sdk :: serde :: { Serialize ,

Deserialize } ;

/// Main contract structure serialized with Borsh

[near_bindgen]

[derive(BorshDeserialize, BorshSerialize, PanicOnDefault)]

pub

struct

Contract

{ pub pair :

Pair , }

/// Implements both serde and borsh serialization. /// serde is typically useful when returning a struct in JSON format for a frontend.

[derive(Serialize, Deserialize, BorshDeserialize, BorshSerialize)]

[serde(crate =

"near_sdk::serde")] pub

struct

Pair

{ pub a :

u32 , pub b :

u32 , }

[near_bindgen]

impl

Contract

{

[init]

pub

fn

new (pair :

Pair)

->

Self

{ Self

{ pair , } }

pub

fn

get_pair (self)

->

Pair

{ self . pair } }

std::panic!

vsenv::panic

- std::panic!
- panics the current thread. It usesformat!
- internally, so it can take arguments.
- SDK sets up a panic hook, which converts the generatedPanicInfo
- frompanic!
- into a string and usesenv::panic
- internally to report it to Runtime.
- This may provide extra debugging information such as the line number of the source code where the panic happened.
- env::panic
- directly calls the host method to panic the contract.
- It doesn't provide any other extra debugging information except for the passed message.

Use workspaces

Workspaces allow you to automate workflows and run tests for multiple contracts and cross-contract calls in a sandbox or testnet environment. Read more,workspaces-rs orworkspaces-js . Edit this page Last updatedonAug 24, 2022 byDamián Parrino Was this page helpful? Yes No

Previous Unit Tests Next Reducing Contract Size