Sending NEAR

You might want to send tokens from a contract for many reasons.

- The contract uses something like theStorage Standard
- and needs to return deposits to users when they unregister.
- Users pay into the contract and the contract later pays these fees to the maintainers, redistributes them to users, or disburses them to some cause the users vote on.
- And more!

Blockchains give us programmable money, and the ability for a smart contract to send tokens lies at the heart of that ability.

NEAR makes this easy. Transferring NEAR tokens is the simplest transaction you can send from a smart contract. Here's all you need:

let amount :

u128

=

1_000_000_000_000_000_000_000_000 ;

// 1 NEAR as yoctoNEAR let account_id :

AccountId

=

"example.near" . parse ( ) . unwrap ( ) ;

Promise :: new (account_id) . transfer (amount) ; In the context of a full contract and function call, this could look like:

use

near_sdk :: { json_types :: U128 , near_bindgen ,

AccountId ,

Promise } ;

[near_bindgen]

pub

struct

Contract

{ }

[near_bindgen]

impl

Contract

{ pub

fn

pay (amount :

U128 , to :

AccountId)

->

Promise

{ Promise :: new (to) . transfer (amount .0) } } Most of this is boilerplate you're probably familiar with by now – imports, setting upnear_bindgen ,borsh , etc. Some interesting details related to the transfer itself:

- U128
- with a capitalU
- : Thepay
- method defined here accepts JSON as input, and numbers in JScannot be larger than2^53-1
- , so for compatibility with deserializing JSON to JS, the integer is serialized as a decimal string. Since thetransfer
- method takes a number inyocto
- NEAR, it's likely to need numbers much larger than2^53-1
- .
- When a function takesU128
- as input, it means that callers need to specify the number a a string. near-sdk-rs will then cast it toU128
- type, which wraps Rust's nativeu128
- . The underlyingu128
- can be retrieved with.0
- – used intransfer(amount.0)
- .
- AccountId
- : this will automatically check that the provided string is a well-formed NEAR account ID, and panic with a useful error if not.
- ReturningPromise
- : This allows NEAR Explorer, near-cli, near-api-js, and other tooling to correctly determine if a whole chain of transactions is successful. If your function does not returnPromise
- , tools like near-cli will return immediately after your function call. And then even if thetransfer
- fails, your function call will be considered successful. You can see an example of this behaviorhere
- .

Using near-cli or near-cli-rs, someone could invoke this function with a call like:

- near-cli
- near-cli-rs

near call pay '{"amount": "1000000000000000000000000", "to": "example.near"}' --accountId benjiman.near near contract call-function as-transaction pay json-args '{"amount": "1000000000000000000000000", "to": "example.near"}' prepaid-gas '30 TeraGas' attached-deposit '0 NEAR' sign-as benjiman.near network-config testnet sign-with-keychain send Edit this page Last updatedonMar 12, 2024 bymatiasbenary Was this page helpful? Yes No

Previous Promises: Introduction Next Creating Accounts