The linkdrop contract

We're going to take a small detour to talk about the linkdrop smart contract. It's best that we first understand this contract and its purpose, then discuss calling a method on this contract.

The linkdrop contract is deployed to the accountstestnet andnear , which are known as the top-level accounts of the testnet and mainnet network, respectively. (Anyone can create a linkdrop-style contract elsewhere, but the one shown here is the main one that others are patterned off of.)

Testnet

There’s nothing special about testnet accounts; there is no real-world cost to you as a developer when creating testnet accounts, so feel free to create or delete at your convenience.

When a user signs up for a testnet account on NEAR Wallet, they'll see this:

Let's discuss how this testnet account gets created.

Notice the new account will end in.testnet . This is because the accounttestnet will create a subaccount (like we learned aboutearlier in this tutorial) calledvacant-name.testnet .

There are two ways to create this subaccount:

1. Use a full-access key for the accounttestnet
2. to sign a transaction with theCreateAccount
3. Action.
4. In a smart contract deployed to thetestnet
5. account, call theCreateAccount
6. Action, which is an async method that returns a Promise. (More info about writing aCreateAccount Promise
7. .)

(In the example below that uses NEAR CLI to create a new account, it's callingCreateAccount on the linkdrop contract that is deployed to the top level "near" account on mainnet.)

Mainnet

On mainnet, the accountnear also has the linkdrop contract deployed to it.

Using NEAR CLI, a person can create a mainnet account by calling the linkdrop contract, like shown below:

The above command calls thecreate_account method on the accountnear , and would createaloha.near if it's available , funding it with 15 Ⓝ.

We'll want to write a smart contract that calls that same method. However, things get interesting because it's possiblealoha.near is already taken, so we'll need to learn how to handle that.

A simple callback

Thecreate_account

method

Here, we'll show the implementation of thecreate_account method. Note the#[payable] macro, which allows this function to accept an attached deposit. (Remember in the CLI command we were attaching 15 Ⓝ.)

src/lib.rs loading ... See full example on GitHub The most important part of the snippet above is around the middle where there's:

Promise::new(...) ... .then(Self::ext(env::current_account_id()) .on_account_created(...)) This translates to, "we're going to attempt to perform an Action, and when we're done, please call myself at the methodon_account_created so we can see how that went."

This doesn't work Not infrequently, developers will attempt to do this in a smart contract:

let creation_result =

Promise :: new ("aloha.mike.near") . create_account ( ) ;

// Check creation_result variable (can't do it!) if creation_result { ... } In other programming languages promises might work like this, but we must use callbacks instead.

The callback

Now let's look at the callback:

src/lib.rs loading ... See full example on GitHub This calls the private helper methodis_promise_success , which basically checks to see that there was only one promise result, because we only attempted one Promise:

src/lib.rs loading ... See full example on GitHub Note that the callback returns a boolean. This means when we modify our crossword puzzle to call the linkdrop contract ontestnet , we'll be able to determine if the account creation succeeded or failed.

And that's it! Now we've seen a method and a callback in action for a simple contract.

This is important Understanding cross-contract calls and callbacks is quite important in smart contract development.

Since NEAR's transactions are asynchronous, the use of callbacks may be a new paradigm shift for smart contract developers from other ecosystems.

Feel free to dig into the linkdrop contract and play with the ideas presented in this section.

There are two additional examples that are helpful to look at:

1. High-level cross-contract calls
2. — this is similar what we've seen in the linkdrop contract.
3. Low-level cross-contract calls
4. — a different approach where you don't use the traits we mentioned.

Next we'll modify the crossword puzzle contract to check for the signer's public key, which is how we now determine if they solved the puzzle correctly. Edit this page Last updatedonJan 19, 2024 byDamián Parrino Was this page helpful? Yes No

Previous Seed phrase logic Next Cross-contract calls, etc.