# Transferring Fungible Tokens

In this tutorial, you'll learn how to implement thecore standards into your smart contract. You'll implement the logic that allows you to transfer and receive tokens. If you're joining us for the first time, feel free to clonethis repository and follow along in the4.storage folder.

tip If you wish to see the finished code for thisCore tutorial, you can find it in the5.transfers folder.

## Introduction

Up until this point, you've created a simple FT smart contract that allows the owner to mint a total supply of tokens and view information about the Fungible Token itself. In addition, you've added the functionality to register accounts and emit events. Today, you'll expand your smart contract to allow for users to transfer and receive fungible tokens.

The logic for doing a simple transfer is quite easy to understand. Let's say Benji wants to transfer Mike 100 of his fungible tokens. The contract should do a few things:

- Check if Benji owns at least 100 tokens.
- Make sure Benji is calling the function.
- Ensure Mike is registered on the contract.
- Take 100 tokens out of Benji's account.
- Put 100 tokens into Mike's account.

At this point, you're ready to move on and make the necessary modifications to your smart contract.

## Modifications to the contract

Let's start our journey in thesrc/ft_core.rs file.

### Transfer function

You'll start by implementing theft_transfer logic which is found in thesrc/ft_core.rs file. This function will transfer the specifiedamount to thereceiver_id with an optionalmemo such as"Happy Birthday Mike!" .

5.transfers/src/ft_core.rs loading ... See full example on GitHub There are a couple things to notice here.

1. We've introduced a new function calledassert_one_yocto()
2. . This method will ensure that the user is signing the transaction with a full access key by requiring a deposit of exactly 1 yoctoNEAR, the smallest possible amount of NEAR that can be transferred. Since the transfer function is potentially transferring very valuable assets, you'll want to make sure that whoever is calling the function has a full access key.
3. We've introduced aninternal_transfer
4. method. This will perform all the logic necessary to transfer the tokens internally.

### Internal helper functions

Let's quickly move over to theft-contract/src/internal.rs file so that you can implement theinternal_transfer method which is the core of this tutorial. This function will take the following parameters:

- sender_id
- : the account that's attempting to transfer the tokens.
- receiver_id
- : the account that's receiving the tokens.
- amount
- : the amount of FTs being transferred.
- memo
- : an optional memo to include.

The first thing you'll want to do is make sure the sender isn't sending tokens to themselves and that they're sending a positive number. After that, you'll want to withdraw the tokens from the sender's balance and deposit them into the receiver's balance. You've already written the logic to deposit FTs by using theinternal_deposit function.

Let's use similar logic to implementinternal_withdraw :

5.transfers/src/internal.rs loading ... See full example on GitHub In this case, the contract will get the account's balance and ensure they are registered by calling theinternal_unwrap_balance_of function. It will then subtract the amount from the user's balance and re-insert them into the map.

Using theinternal_deposit andinternal_withdraw functions together, the core of theinternal_transfer function is complete.

There's only one more thing you need to do. When transferring the tokens, you need to remember to emit a log as per theevents standard:

5.transfers/src/internal.rs loading ... See full example on GitHub Now that this is finished, the simple transfer case is done! You can now transfer FTs between registered users!

### Transfer call function

In this section, you'll learn about a new functionft_transfer_call . This will transfer FTs to a receiver and also call a method on the receiver's contract all in the same transaction.

Let's consider the following scenario. An account wants to transfer FTs to a smart contract for performing a service. The traditional approach would be to perform the service and then ask for the tokens in two separate transactions. If we introduce a “transfer and call” workflow baked into a single transaction, the process can be greatly improved:

5.transfers/src/ft_core.rs loading ... See full example on GitHub This function will do several things:

1. Ensures the caller attached exactly 1 yocto for security purposes.
2. Transfer the tokens using theinternal_transfer
3. you just wrote.
4. Creates a promise to call the methodft_on_transfer
5. on thereceiver_id
6. 's contract.
7. After the promise finishes executing, the functionft_resolve_transfer
8. is called.

info This is a very common workflow when dealing with cross contract calls. You first initiate the call and wait for it to finish executing. Then, you invoke a function that resolves the result of the promise and act accordingly.

Learn morehere . When callingft_on_transfer , it will return how many tokens the contract should refund the original sender.

This is important for a couple of reasons:

1. If you send the receiver too many FTs and their contract wants to refund the excess.
2. If any of the logic panics, all of the tokens should be refunded back to the sender.

This logic will all happen in theft_resolve_transfer function:

5.transfers/src/ft_core.rs loading ... See full example on GitHub The first thing you'll do is check the status of the promise. If anything failed, you'll refund the sender for the full amount of tokens. If the promise succeeded, you'll extract the amount of tokens to refund the sender based on the value returned fromft_on_transfer . Once you have the amount needed to be refunded, you'll perform the actual refund logic by using theinternal_transfer function you wrote previously.

You'll then return the net amount of tokens that were transferred to the receiver. If the sender transferred 100 tokens but 20 were refunded, this function should return 80.

With that finished, you've now successfully added the necessary logic to allow users to transfer FTs. It's now time to deploy and do some testing.

## Deploying the contract

Let's create a new sub-account to deploy the contract to. Since these changes are only additive and non state breaking, you could have deployed a patch fix to the contract you deployed in the storage section as well. To learn more about upgrading contracts, see theupgrading a contract section in the NFT Zero to Hero tutorial.

### Creating a sub-account

Run the following command to create a sub-accounttransfer of your main account with an initial balance of 25 NEAR which will be transferred from the original to your new account.

near create-account transfer.FT_CONTRACT_ID --masterAccount FT_CONTRACT_ID --initialBalance 25 Next, you'll want to export an environment variable for ease of development:

export TRANSFER_FT_CONTRACT_ID=transfer.FT_CONTRACT_ID Using the build script, build the deploy the contract as you did in the previous tutorials:

cd 1.skeleton && ./build.sh && cd .. && near deploy --wasmFile out/contract.wasm --accountId TRANSFER_FT_CONTRACT_ID tip If you haven't completed the previous tutorials and are just following along with this one, simply create an account and login with your CLI usingnear login . You can then export an environment variableexport TRANSFER_FT_CONTRACT_ID=YOUR_ACCOUNT_ID_HERE . In addition, you can find the contract code by going to

the5.transfers folder. Instead of building using1.skeleton , you can build by going to the5.transfers folder and running./build.sh .

## Initialization

Now that the contract is deployed, it's time to initialize it and mint the total supply. Let's once again create an initial supply of 1000gtNEAR .

near call TRANSFER_FT_CONTRACT_ID new_default_meta '{"owner_id": "'TRANSFER_FT_CONTRACT_ID'", "total_supply": "1000000000000000000000000000"}' --accountId TRANSFER_FT_CONTRACT_ID You can check if you own the FTs by running the following command:

near view TRANSFER_FT_CONTRACT_ID ft_balance_of '{"account_id": "'TRANSFER_FT_CONTRACT_ID'"}'

## Testing the transfer function

Let's test the transfer function by transferring 1gtNEAR from the owner account to the accountbenjiman.testnet

note The FTs won't be recoverable unless the accountbenjiman.testnet transfers them back to you. If you don't want your FTs lost, make a new account and transfer the token to that account instead. You'll first need to register the accountbenjiman.testnet by running the following command.

near call TRANSFER_FT_CONTRACT_ID storage_deposit '{"account_id": "benjiman.testnet"}' --accountId TRANSFER_FT_CONTRACT_ID --amount 0.01 Once the account is registered, you can transfer the FTs by running the following command. Take note that you're also attaching exactly 1 yoctoNEAR by using the--depositYocto flag.

near call TRANSFER_FT_CONTRACT_ID ft_transfer '{"receiver_id": "benjiman.testnet", "amount": "1000000000000000000000000", "memo": "Go Team :)"}' --accountId TRANSFER_FT_CONTRACT_ID --depositYocto 1 You should see theFtTransferEvent being emitted in the console. At this point, if you check for the total supply, it should still be 1000gtNEAR but if you check both the balance of Benji and the balance of the owner, they should reflect the transfer.

near view TRANSFER_FT_CONTRACT_ID ft_balance_of '{"account_id": "'TRANSFER_FT_CONTRACT_ID'"}' near view TRANSFER_FT_CONTRACT_ID ft_balance_of '{"account_id": "benjiman.testnet"}' near view TRANSFER_FT_CONTRACT_ID ft_total_supply

## Testing the transfer call function

Now that you've tested theft_transfer function, it's time to test theft_transfer_call function. If you try to transfer tokens to a receiver that doesnot implement theft_on_transfer function, the contract will panic and the FTs will berefunded . Let's test this functionality below.

You can try to transfer the FTs to the accountno-contract.testnet which, as the name suggests, doesn't have a contract. This means that the receiver doesn't implement theft_on_transfer function and the FTs should remain yours after the transaction is complete. You'll first have to register the account, however.

near call TRANSFER_FT_CONTRACT_ID storage_deposit '{"account_id": "no-contract.testnet"}' --accountId TRANSFER_FT_CONTRACT_ID --amount 0.01 near call TRANSFER_FT_CONTRACT_ID ft_transfer_call '{"receiver_id": "no-contract.testnet", "amount": "1000000000000000000000000", "msg": "foo"}' --accountId TRANSFER_FT_CONTRACT_ID --depositYocto 1 --gas 200000000000000 The output response should be as follows.

Scheduling a call: transfer.dev-1660680326316-91393402417293.ft_transfer_call({"receiver_id": "no-contract.testnet", "amount": "1000000000000000000000000", "msg": "foo"}) with attached 0.000000000000000000000001 NEAR Doing account.functionCall() Receipts: AJ3yWv7tSiZRLtoTkyTgfdzmQP1dpjX9DxJDiD33uwTZ, EKtpDFoJWNnbyxJ7SriAFQYX8XV9ZTzwmCF2qhSaYMAc, 21UzDZ791pWZRKAHv8WaRKN8mqDVrz8zewLWGTNZTckh Log [transfer.dev-1660680326316-91393402417293]: EVENT_JSON: {"standard":"nep141","version":"1.0.0","event":"ft_transfer","data":[{"old_owner_id":"transfer.dev-1660680326316-91393402417293","new_owner_id":"no-contract.testnet","amount":"1000000000000000000000000"}]} Receipt: 5N2WV8picxwUNC5TYa3A3v4qGquQAhkU6P81tgRt1UFN Failure [transfer.dev-1660680326316-91393402417293]: Error: Cannot find contract code for account no-contract.testnet Receipt: AdT1bSZNCu9ACq7m6ynb12GgSb3zBenfzvvzRwfYPBmg Log [transfer.dev-1660680326316-91393402417293]: EVENT_JSON:{"standard":"nep141","version":"1.0.0","event":"ft_transfer","data":[{"old_owner_id":"no-contract.testnet","new_owner_id":"transfer.dev-1660680326316-91393402417293","amount":"1000000000000000000000000","memo":"Refund"}]} Transaction Id 2XVy8MZU8TWreW8C9zK6HSyBsxE5hyTbyUyxNdncxL8g To see the transaction in the transaction explorer, please open this url in your browser https://testnet.nearblocks.io/txns/2XVy8MZU8TWreW8C9zK6HSyBsxE5hyTbyUyxNdncxL8g '0' There should be a transfer event emitted for the initial transfer of tokens and then also for the refund. In addition,0 should have been returned from the function because the sender ended up transferring net 0 tokens to the receiver since all the tokens were refunded.

If you query for the balance ofno-contract.testnet , it should still be 0.

near view TRANSFER_FT_CONTRACT_ID ft_balance_of '{"account_id": "no-contract.testnet"}' Hurray! At this point, your FT contract is fully complete and all the functionality is working as expected. Go forth and experiment! The world is your oyster and don't forget, go team!

## Conclusion

In this tutorial, you learned how to expand a FT contract by adding ways for users to transfer FTs. You broke down the problem into smaller, more digestible subtasks and took that information and implemented both the FT transfer and FT transfer call functions. In addition, you deployed another contract and tested the transfer functionality.

In the next tutorial , you'll learn about how an NFT marketplace can operate to purchase NFTs by using Fungible Tokens. Edit this page Last updated on Feb 9, 2024 by gagdiez Was this page helpful? Yes No

Previous Registering Accounts Next Adding FTs to a Marketplace