

Core

In this tutorial you'll learn how to implement the [core standards](#) into your smart contract. If you're joining us for the first time, feel free to clone [this repo](#) and checkout the `3.enumeration` branch to follow along.

git checkout 3.enumeration tip If you wish to see the finished code for this Core tutorial, you can find it on the `4.core` branch.

Introduction

Up until this point, you've created a simple NFT smart contract that allows users to mint tokens and view information using the [enumeration standards](#). Today, you'll expand your smart contract to allow for users to not only mint tokens, but transfer them as well.

As we did in the [minting tutorial](#), let's break down the problem into multiple subtasks to make our lives easier. When a token is minted, information is stored in 3 places:

- `tokens_per_owner`
- : set of tokens for each account.
- `tokens_by_id`
- : maps a token ID to a `Token` object.
- `token_metadata_by_id`
- : maps a token ID to its metadata.

Let's now consider the following scenario. If Benji owns token A and wants to transfer it to Mike as a birthday gift, what should happen? First of all, token A should be removed from Benji's set of tokens and added to Mike's set of tokens.

If that's the only logic you implement, you'll run into some problems. If you were to do a view call to query for information about that token after it's been transferred to Mike, it would still say that Benji is the owner.

This is because the contract is still mapping the token ID to the old `Token` object that contains the `owner_id` field set to Benji's account ID. You still have to change the `tokens_by_id` data structure so that the token ID maps to a new `Token` object which has Mike as the owner.

With that being said, the final process for when an owner transfers a token to a receiver should be the following:

- Remove the token from the owner's set.
- Add the token to the receiver's set.
- Map a token ID to a new `Token` object containing the correct owner.

note You might be curious as to why we don't edit the `token_metadata_by_id` field. This is because no matter who owns the token, the token ID will always map to the same metadata. The metadata should never change and so we can just leave it alone. 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 the `nft-contract/src/nft_core.rs` file.

Transfer function

You'll start by implementing the `nft_transfer` logic. This function will transfer the specified `token_id` to the `receiver_id` with an optional memo such as "Happy Birthday Mike!".

`nft-contract/src/nft_core.rs` loading ... [See full example on GitHub](#) There are a couple things to notice here. Firstly, we've introduced a new function called `assert_one_yocto()`. This method will ensure that the user has attached exactly one `yoctoNEAR` to the call. If a function requires a deposit, you need a full access key to sign that transaction. By adding the one `yoctoNEAR` deposit requirement, you're essentially forcing the user to sign the transaction with a full access key.

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.

Secondly, we've introduced an `internal_transfer` method. This will perform all the logic necessary to transfer an NFT.

Internal helper functions

Let's quickly move over to the `nft-contract/src/internal.rs` file so that you can implement the `assert_one_yocto()` and `internal_transfer` methods.

Let's start with the easier one, `assert_one_yocto()` .

assert_one_yocto

You can put this function anywhere in the `internal.rs` file but in our case, we'll put it after the `hash_account_id` function:

nft-contract/src/internal.rs loading ... [See full example on GitHub](#)

internal_transfer

It's now time to implement the `internal_transfer` function which is the core of this tutorial. This function will take the following parameters:

- `sender_id`
 - : the account that's attempting to transfer the token.
- `receiver_id`
 - : the account that's receiving the token.
- `token_id`
 - : the token ID being transferred.
- `memo`
 - : an optional memo to include.

The first thing you'll want to do is to make sure that the sender is authorized to transfer the token. In this case, you'll just make sure that the sender is the owner of the token. You'll do that by getting the `Token` object using the `token_id` and making sure that the sender is equal to the token's `sowner_id` .

Second, you'll remove the token ID from the sender's list and add the token ID to the receiver's list of tokens. Finally, you'll create a new `Token` object with the receiver as the owner and remap the token ID to that newly created object.

You'll want to create this function within the contract implementation (below the `internal_add_token_to_owner` you created in the minting tutorial).

nft-contract/src/internal.rs loading ... [See full example on GitHub](#) You've previously implemented functionality for adding a token ID to an owner's set but you haven't created the functionality for removing a token ID from an owner's set. Let's do that now by creating a new function called `internal_remove_token_from_owner` which we'll place right above our `internal_transfer` and below the `internal_add_token_to_owner` function.

In the remove function, you'll get the set of tokens for a given account ID and then remove the passed in token ID. If the account's set is empty after the removal, you'll simply remove the account from the `tokens_per_owner` data structure.

nft-contract/src/internal.rs loading ... [See full example on GitHub](#) Your `internal.rs` file should now have the following outline:

internal.rs |—— hash_account_id |—— assert_one_yocto |—— refund_deposit |—— impl Contract |——
internal_add_token_to_owner |—— internal_remove_token_from_owner |—— internal_transfer With these internal functions complete, the logic for transferring NFTs is finished. It's now time to move on and implement `nft_transfer_call` , one of the most integral yet complicated functions of the standard.

Transfer call function

Let's consider the following scenario. An account wants to transfer an NFT to a smart contract for performing a service. The traditional approach would be to use an approval management system, where the receiving contract is granted the ability to transfer the NFT to themselves after completion. You'll learn more about the approval management system in the [approvals section](#) of the tutorial series.

This allowance workflow takes multiple transactions. If we introduce a “transfer and call” workflow baked into a single transaction, the process can be greatly improved.

For this reason, we have a function `nft_transfer_call` which will transfer an NFT to a receiver and also call a method on the receiver's contract all in the same transaction.

nft-contract/src/nft_core.rs loading ... [See full example on GitHub](#) The function will first assert that the caller attached exactly 1 yocto for security purposes. It will then transfer the NFT using `internal_transfer` and start the cross contract call. It will call the method `nft_on_transfer` on the `receiver_id`'s contract which returns a promise. After the promise finishes executing, the function `nft_resolve_transfer` is called. This is a very common workflow when dealing with cross contract calls. You first initiate the call and wait for it to finish executing. You then invoke a function that resolves the result of the promise and act accordingly.

In our case, when calling `nft_on_transfer` , that function will return whether or not you should return the NFT to its original owner in the form of a boolean. This logic will be executed in the `nft_resolve_transfer` function.

nft-contract/src/nft_core.rs loading ... [See full example on GitHub](#) If `ifnft_on_transfer` returned true, you should send the token back to its original owner. On the contrary, if false was returned, no extra logic is needed. As for the return value `ofnft_resolve_transfer`, the standard dictates that the function should return a boolean indicating whether or not the receiver successfully received the token or not.

This means that if `ifnft_on_transfer` returned true, you should return false. This is because if the token is being returned its original owner, the receiver_id didn't successfully receive the token in the end. On the contrary, if `ifnft_on_transfer` returned false, you should return true since we don't need to return the token and thus the receiver_id successfully owns the token.

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

Redeploying the contract

Using the build script, build and deploy the contract as you did in the previous tutorials:

`yarn build && near deploy NFT_CONTRACT_ID out/main.wasm` This should output a warning saying that the account has a deployed contract and will ask if you'd like to proceed. Simply type y and hit enter.

This account already has a deployed contract [AKJK7sCysrWrFZ976YVBnm6yzmJuKLzdAyssfzK9yLsa]. Do you want to proceed? (y/n) 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 using `near login`. You can then export an environment variable `export NFT_CONTRACT_ID=YOUR_ACCOUNT_ID_HERE`.

Testing the new changes

Now that you've deployed a patch fix to the contract, it's time to move onto testing. Using the previous NFT contract where you had minted a token to yourself, you can test the `thenft_transfer` method. If you transfer the NFT, it should be removed from your account's collectibles displayed in the wallet. In addition, if you query any of the enumeration functions, it should show that you are no longer the owner.

Let's test this out by transferring an NFT to the account `benjiman.testnet` and seeing if the NFT is no longer owned by you.

Testing the transfer function

note This means that the NFT won't be recoverable unless the account `benjiman.testnet` transfers it back to you. If you don't want your NFT lost, make a new account and transfer the token to that account instead. If you run the following command, it will transfer the token "token-1" to the account `benjiman.testnet` with the memo "Go Team :)". Take note that you're also attaching exactly 1 yoctoNEAR by using the `--depositYocto` flag.

tip If you used a different token ID in the previous tutorials, replace `token-1` with your token ID. `near call NFT_CONTRACT_ID nft_transfer '{"receiver_id": "benjiman.testnet", "token_id": "token-1", "memo": "Go Team :)"}' --accountId NFT_CONTRACT_ID --depositYocto 1` If you now query for all the tokens owned by your account, that token should be missing. Similarly, if you query for the list of tokens owned by `benjiman.testnet`, that account should now own your NFT.

Testing the transfer call function

Now that you've tested the `thenft_transfer` function, it's time to test the `thenft_transfer_call` function. If you try to transfer an NFT to a receiver that does not implement the `thenft_on_transfer` function, the contract will panic and the NFT will not be transferred. Let's test this functionality below.

First mint a new NFT that will be used to test the transfer call functionality.

```
near call NFT_CONTRACT_ID nft_mint '{"token_id": "token-2", "metadata": {"title": "NFT Tutorial Token", "description": "Testing the transfer call function", "media": "https://bafybeifcztwrt3k7a2k4vutd3amkwsmaqyhrd3lhvpt33dyjvufusq.ipfs.dweb.link/goteam-gif.gif"}', "receiver_id": "NFT_CONTRACT_ID"}' --accountId NFT_CONTRACT_ID --amount 0.1
```

Now that you've minted the token, you can try to transfer the NFT to the account `no-contract.testnet` which as the name suggests, doesn't have a contract. This means that the receiver doesn't implement the `thenft_on_transfer` function and the NFT should remain yours after the transaction is complete.

```
near call NFT_CONTRACT_ID nft_transfer_call '{"receiver_id": "no-contract.testnet", "token_id": "token-2", "msg": "foo"}' --accountId NFT_CONTRACT_ID --depositYocto 1 --gas 200000000000000
```

If you query for your tokens, you should still have `token-2` and at this point, you're finished!

Conclusion

In this tutorial, you learned how to expand an NFT contract past the minting functionality and you added ways for users to transfer NFTs. You [broke down](#) the problem into smaller, more digestible subtasks and took that information and implemented both the [NFT transfer](#) and [NFT transfer call](#) functions. In addition, you deployed another [patch fix](#) to your smart contract and [tested](#) the transfer functionality.

In the [next tutorial](#) , you'll learn about the approval management system and how you can approve others to transfer tokens on your behalf.

Versioning for this article At the time of this writing, this example works with the following versions:

- near-cli:4.0.4
- NFT standard:[NEP171](#)
- , version1.1.0
- Enumeration standard:[NEP181](#)
- , version1.0.0 [Edit this page](#) Last updated on Feb 16, 2024 by garikbesson Was this page helpful? Yes No

[Previous Enumeration](#) [Next Approvals](#)