Core

In this tutorial you'll learn how to implement thecore standards into your smart contract. If you're joining us for the first time, feel free to clonethis repo and checkout the3.enumeration branch to follow along.

git checkout 3.enumeration tip If you wish to see the finished code for thisCore tutorial, you can find it on the4.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 theenumeration 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 theminting 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 aToken
- 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 aview 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 oldToken object that contains theowner_id field set to Benji's account ID. You still have to change thetokens_by_id data structure so that the token ID maps to a newToken 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 newToken
- object containing the correct owner.

note You might be curious as to why we don't edit thetoken_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 thenft-contract/src/nft_core.rs file.

Transfer function

You'll start by implementing thenft_transfer logic. This function will transfer the specifiedtoken_id to thereceiver_id with an optionalmemo such as"Happy Birthday Mike!" .

nft-contract/src/nft_core.rs loading ... See full example on GitHubThere are a couple things to notice here. Firstly, we've introduced a new function calledassert_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 aninternal_transfer method. This will perform all the logic necessary to transfer an NFT.

Internal helper functions

Let's quickly move over to thenft-contract/src/internal.rs file so that you can implement theassert_one_yocto() andinternal_transfer methods.

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

assert_one_yocto

You can put this function anywhere in theinternal.rs file but in our case, we'll put it after thehash_account_id function:

nft-contract/src/internal.rs loading ... See full example on GitHub

internal_transfer

It's now time to implement theinternal_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 theToken object using thetoken_id and making sure that the sender is equal to the token'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 newToken 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 theinternal_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 created a new function calledinternal_remove_token_from_owner which we'll place right above ourinternal_transfer and below theinternal_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 thetokens_per_owner data structure.

nft-contract/src/internal.rs loading ... See full example on GitHub Yourinternal.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 implementnft_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 theapprovals 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 functionnft_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 usinginternal_transfer and start the cross contract call. It will call the methodnft_on_transfer on thereceiver_id 's contract which returns a promise. After the promise finishes executing, the functionnft_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 callingnft_on_transfer , that function will return whether or not you should return the NFT to it's original owner in the form of a boolean. This is logic will be executed in thenft_resolve_transfer function.

nft-contract/src/nft_core.rs loading ... See full example on GitHub Ifnft_on_transfer returned true, you should send the token back to it's 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 ifnft_on_transfer returned true, you should return false. This is because if the token is being returned its original owner, thereceiver_id didn't successfully receive the token in the end. On the contrary, ifnft_on_transfer returned false, you should return true since we don't need to return the token and thus thereceiver_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 typey 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 usingnear login . You can then export an environment variableexport 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 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 accountbenjiman.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 accountbenjiman.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 accountbenjiman.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, replacetoken-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 bybenjiman.testnet , that account should now own your NFT.

Testing the transfer call function

Now that you've tested thenft_transfer function, it's time to test thenft_transfer_call function. If you try to transfer an NFT to a receiver that doesnot implement thenft_on_transfer function, the contract will panic and the NFT willnot 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://bafybeiftczwrtyr3k7a2k4vutd3amkwsmaqyhrdzlhvpt33dyjivufqusq.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 accountno-contract.testnet which as the name suggests, doesn't have a contract. This means that the receiver doesn't implement 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 havetoken-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. Youbroke down the problem into smaller, more digestible subtasks and took that information and implemented both theNFT transfer andNFT transfer call functions. In addition, you deployed anotherpatch fix to your smart contract andtested the transfer functionality.

In thenext 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 updatedonFeb 16, 2024 bygarikbesson Was this page helpful? Yes No

Previous Enumeration Next Approvals