# Enumeration

In the previous tutorials, you looked at ways to integrate the minting functionality into a skeleton smart contract. In order to get your NFTs to show in the wallet, you also had to deploy a patch fix that implemented one of the enumeration methods. In this tutorial, you'll expand on and finish the rest of the enumeration methods as per thestandard Now you'll extend the NFT smart contract and add a couple of enumeration methods that can be used to return the contract's state.

## Introduction

As mentioned in theUpgrade a Contract tutorial, you can deploy patches and fixes to smart contracts. This time, you'll use that knowledge to implement thenft_total_supply ,nft_tokens andnft_supply_for_owner enumeration functions.

To get started, either switch to the2.minting branch from ourGitHub repository , or continue your work from the previous tutorials. If you haven't cloned it yet, refer to theContract Architecture to check out the repository.

git checkout 2.minting tip If you wish to see the finished code for thisEnumeration tutorial, you can find it on the3.enumeration branch.

## Modifications to the contract

Let's start by opening thesrc/enumeration.rs file and locating the emptynft_total_supply function.

nft_total_supply

This function should return the total number of NFTs stored on the contract. You can easily achieve this functionality by simply returning the length of thenft_metadata_by_id data structure.

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

This function should return a paginated list ofJsonTokens that are stored on the contract regardless of their owners. If the user provides afrom_index parameter, you should use that as the starting point for which to start iterating through tokens; otherwise it should start from the beginning. Likewise, if the user provides alimit parameter, the function shall stop after reaching either the limit or the end of the list.

tip Rust has useful methods for pagination, allowing you to skip to a starting index and taking the firstn elements of an iterator. nft-contract/src/enumeration.rs loading ... See full example on GitHub nft_supply_for_owner

This function should look for all the non-fungible tokens for a user-defined owner, and return the length of the resulting set. If there isn't a set of tokens for the providedAccountID , then the function shall return0 .

nft-contract/src/enumeration.rs loading ... See full example on GitHub Next, you can use the CLI to query these new methods and validate that they work correctly.

## Redeploying the contract

Now that you've implemented the necessary logic fornft_tokens_for_owner , it's time to build and re-deploy the contract to your account. Using the build script, 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)

## Enumerating tokens

Once the updated contract has been redeployed, you can test and see if these new functions work as expected.

### NFT tokens

Let's query for a list of non-fungible tokens on the contract. Use the following command to query for the information of up to 50 NFTs starting from the 10th item:

near view NFT_CONTRACT_ID nft_tokens '{"from_index": "10", "limit": 50}' This command should return an output similar to the following:

Example response: []

### Tokens by owner

To get the total supply of NFTs owned by thegoteam.testnet account, call thenft_supply_for_owner function and set theaccount_id parameter:

near view NFT_CONTRACT_ID nft_supply_for_owner '{"account_id": "goteam.testnet"}' This should return an output similar to the following:

Example response: 0

## Conclusion

In this tutorial, you have added twonew enumeration functions , and now you have a basic NFT smart contract with minting and enumeration methods in place. After implementing these modifications, you redeployed the smart contract and tested the functions using the CLI.

In thenext tutorial , you'll implement the core functions needed to allow users to transfer the minted tokens.

Remember If you want to see the finished code from this tutorial, you can checkout the3.enumeration branch. 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 Upgrade a Contract Next Core