# 661.https://stackoverflow.com/questions/67543769/erc721-tokens-with-metadata-and-enumerable-with-openzeppelin-v4-1-0

**T:**ERC721 Tokens with Metadata and Enumerable with Openzeppelin v4.1.0

**Q:**My question has two parts,  
  
I am trying to create an ERC721 token using the Openzeppelin contracts with metadata and that is enumerable. My understanding is after openzeppelin v4.0.0 they removed the ERC721Full.sol contract which included metadata and enummerable. I want to use solidity 0.8.0 and so those old contracts wont work, right? When importing and inheriting ERC721Enumerable.sol into the ERC721.sol contract, I get TypeError: Definition of base has to precede definition of derived contractI tried just importing ERC721Enumerable.sol in my own contract, but still errors. I also tried importing the older ERC721Full.sol contract and changing all the pragma 0.5.0 to pragma 0.8.0, but it inherits like a dozen other contracts and changing all of them doesn't seem wise. I tried the same with IERC721Enumerable.sol, still errors. Any ideas? Any help would be amazing!  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Second part. What's the difference between ERC\_\_ and IERC\_\_? What's the purpose of IERC contracts?  
  
Thanks!!  
  
Here's my contract (I'm following a tutorial). I import the regular ERC721 contract, inherit it. it gives me an error when I test and call the totalSupply function because there is no totalSupply function:  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.0;import "@openzeppelin/contracts/token/ERC721/ERC721.sol";contract Color is ERC721 { string[] public colors; mapping(string => bool) \_colorExists; constructor() ERC721("Color", "COLOR") { } function mint(string memory \_color) public { colors.push(\_color); uint \_id = colors.length; \_mint(msg.sender, \_id); \_colorExists[\_color] = true; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
my test script:  
  
const Color = artifacts.require('./Color.sol')require('chai') .use(require('chai-as-promised')) .should()contract('Color', (accounts) => { let contract before(async () => { contract = await Color.deployed() }) describe('deployment', async () => { it('deploys successfully', async () => { contract = await Color.deployed() const address = contract.address console.log(address) assert.notEqual(address, 0x0) assert.notEqual(address,'') assert.notEqual(address, null) assert.notEqual(address, undefined) }) it('has a name', async () => { const name = await contract.name() assert.equal(name, 'Color') }) it('has a symbol', async () => { const symbol = await contract.symbol() assert.equal(symbol, 'COLOR') }) }) describe('minting', async () => { it('creates a new token', async () => { const result = await contract.mint('#00CD22') const totalSupply = await contract.totalSupply() // SUCCESS asert.equal(totalSupply, 1) }) })})  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
this is my error without the enumerable contract/without totalSupply   
  
I can paste the openzeppelin contracts if you like, or link them here  
  
I also tried this, importing ERC721Enumerable  
  
And got this:  
  
let me know fi you need anymore info!thanks in advance

**C1:**Please edit your question and show the contract code that we can copy-paste into Remix (or any IDE) and it would throw the same error. I was not able to find combination of situations that would simulate it from just the description.

**C2:**I'm having this same exact issue! How do we use the @openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol contract? How do we call it? I'm trying to use the totalSupply function from it.

2 **Answer**

**A1:**An ERC721 by default does not have a totalSupply method, and that's the reason you're getting the error. The totalSupply method comes from the IERC721Enumerable, wich is an optional extension of the standard ERC721 as the documentation states. If you want your ERC721 to be enumerable just import the enumerable extension from the openzeppelin implementations for your derived contract, like so:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.0;import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";import "@openzeppelin/contracts/token/ERC721/ERC721.sol";contract Color is ERC721Enumerable { string[] public colors; mapping(string => bool) \_colorExists; constructor() ERC721("Color", "COLOR") { } function mint(string memory \_color) public { colors.push(\_color); uint \_id = colors.length; \_mint(msg.sender, \_id); \_colorExists[\_color] = true; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
The reason the compiler gave you the error when trying to import ERC721Enumerable is that you were tryning to import it in the Openzeppelin ERC721 implementation, but that contract had to exists prior to the ERC721Enumberable. In other words the inheritance chain is  
  
ERC721 <-- ERC721Enumerable  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
What you were trying to do was  
  
ERC721 <-- ERC721Enumerable |\_\_\_\_\_\_\_\_\_\_\_\_\_↑  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Wich creates a circular dependency that cannot be satisfied.  
  
ERC contracts are like abstracts classes in every OOP programming languages (first that come to my mind and maybe most relatable are Java and C++), while IERC are interfaces; this means that while both cannot be instanciated directly (they both need the children to implement something) ERC contracts provide standard implementations for the corresponding IERC methods. This is the reason often you see contracts implement the ERC contracts and not the IERC ones.

**A2:**To use the ERC271Enumerable extension you need to implement it and override some functions of ERC271, \_beforeTokenTransfer and supportsInterface.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
// SPDX-License-Identifier: MIT pragma solidity ^0.8.0; import "@openzeppelin/contracts/token/ERC721/ERC721.sol";import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";contract Color is ERC721, ERC721Enumerable{ string[] public colors; mapping(string => bool) \_colorExists; constructor () ERC721("Color", "COLORS") {} function \_beforeTokenTransfer(address from, address to, uint256 tokenId) internal override(ERC721, ERC721Enumerable) { super.\_beforeTokenTransfer(from, to, tokenId); } function supportsInterface(bytes4 interfaceId) public view override(ERC721, ERC721Enumerable) returns (bool) { return super.supportsInterface(interfaceId); } function mint(string memory \_color) public{ colors.push(\_color); uint \_id = colors.length; \_mint(msg.sender, \_id); \_colorExists[\_color] = true; }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
What's the difference between ERC\_\_ and IERC\_\_? What's the purpose of IERC contracts?  
  
 ● IERC is the Interface for the token contract.  
  
 ● ERC is the implementation of the token contract.  
  
The importance is to make sure a Contract implementation has the right methods, with the right visibility, parameters, and return values.