# 637.https://stackoverflow.com/questions/68951548/factory-error-the-called-function-should-be-payable

**T:**Factory error The called function should be payable

**Q:**I am using the Opensea Creatures repository (https://github.com/ProjectOpenSea/opensea-creatures.git) to make a Factory and create my ERC721 tokens. I have added the function \_setTokenURI (newTokenId, metadataURI); in ERC721Tradable.sol to add the URI when doing mint but it gives me the following error:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
The transaction has been reverted to the initial state.Note: The called function should be payable if you send value and the value you send should be less than your current balance.Debug the transaction to get more information.   
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
the files involved are these:  
  
CreatureFactory.sol  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.0;import "openzeppelin-solidity/contracts/access/Ownable.sol";import "openzeppelin-solidity/contracts/utils/Strings.sol";import "openzeppelin-solidity/contracts/token/ERC721/ERC721.sol";import "./IFactoryERC721.sol";import "./Creature.sol";contract CreatureFactory is FactoryERC721, Ownable { using Strings for string; event Transfer( address indexed from, address indexed to, uint256 indexed tokenId ); address public proxyRegistryAddress; address public nftAddress; address public lootBoxNftAddress; string public baseURI = "https://terrorverso.herokuapp.com/api/token/"; /\* \* Enforce the existence of only 100 OpenSea creatures. \*/ uint256 CREATURE\_SUPPLY = 100; /\* \* Three different options for minting Creatures (basic, premium, and gold). \*/ uint256 NUM\_OPTIONS = 3; uint256 SINGLE\_CREATURE\_OPTION = 0; uint256 MULTIPLE\_CREATURE\_OPTION = 1; uint256 LOOTBOX\_OPTION = 2; uint256 NUM\_CREATURES\_IN\_MULTIPLE\_CREATURE\_OPTION = 4; constructor(address \_proxyRegistryAddress, address \_nftAddress) { proxyRegistryAddress = \_proxyRegistryAddress; nftAddress = \_nftAddress; fireTransferEvents(address(0), owner()); } function name() override external pure returns (string memory) { return "Factory"; } function symbol() override external pure returns (string memory) { return "FACT"; } function supportsFactoryInterface() override public pure returns (bool) { return true; } function numOptions() override public view returns (uint256) { return NUM\_OPTIONS; } function transferOwnership(address newOwner) override public onlyOwner { address \_prevOwner = owner(); super.transferOwnership(newOwner); fireTransferEvents(\_prevOwner, newOwner); } function fireTransferEvents(address \_from, address \_to) private { for (uint256 i = 0; i < NUM\_OPTIONS; i++) { emit Transfer(\_from, \_to, i); } } function mint(uint256 \_optionId, address \_toAddress, string memory metadataURI) override public { // Must be sent from the owner proxy or owner. ProxyRegistry proxyRegistry = ProxyRegistry(proxyRegistryAddress); assert( address(proxyRegistry.proxies(owner())) == \_msgSender() || owner() == \_msgSender() || \_msgSender() == lootBoxNftAddress ); require(canMint(\_optionId)); Creature openSeaCreature = Creature(nftAddress); if (\_optionId == SINGLE\_CREATURE\_OPTION) { openSeaCreature.mintTo(\_toAddress,metadataURI); } else if (\_optionId == MULTIPLE\_CREATURE\_OPTION) { for ( uint256 i = 0; i < NUM\_CREATURES\_IN\_MULTIPLE\_CREATURE\_OPTION; i++ ) { openSeaCreature.mintTo(\_toAddress,metadataURI); } } } function canMint(uint256 \_optionId) override public view returns (bool) { if (\_optionId >= NUM\_OPTIONS) { return false; } Creature openSeaCreature = Creature(nftAddress); uint256 creatureSupply = openSeaCreature.totalSupply(); uint256 numItemsAllocated = 0; if (\_optionId == SINGLE\_CREATURE\_OPTION) { numItemsAllocated = 1; } else if (\_optionId == MULTIPLE\_CREATURE\_OPTION) { numItemsAllocated = NUM\_CREATURES\_IN\_MULTIPLE\_CREATURE\_OPTION; } return creatureSupply < (CREATURE\_SUPPLY - numItemsAllocated); } function tokenURI(uint256 \_optionId) override external view returns (string memory) { return string(abi.encodePacked(baseURI, Strings.toString(\_optionId))); } /\*\* \* Hack to get things to work automatically on OpenSea. \* Use transferFrom so the frontend doesn't have to worry about different method names. \*/ function transferFrom( address \_from, address \_to, uint256 \_tokenId, string memory metadataURI ) public { mint(\_tokenId, \_to, metadataURI); } /\*\* \* Hack to get things to work automatically on OpenSea. \* Use isApprovedForAll so the frontend doesn't have to worry about different method names. \*/ function isApprovedForAll(address \_owner, address \_operator) public view returns (bool) { if (owner() == \_owner && \_owner == \_operator) { return true; } ProxyRegistry proxyRegistry = ProxyRegistry(proxyRegistryAddress); if ( owner() == \_owner && address(proxyRegistry.proxies(\_owner)) == \_operator ) { return true; } return false; } /\*\* \* Hack to get things to work automatically on OpenSea. \* Use isApprovedForAll so the frontend doesn't have to worry about different method names. \*/ function ownerOf(uint256 \_tokenId) public view returns (address \_owner) { return owner(); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
ERC721Tradable.sol:  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.0;import "openzeppelin-solidity/contracts/token/ERC721/ERC721.sol";import "openzeppelin-solidity/contracts/token/ERC721/extensions/ERC721Enumerable.sol";import "openzeppelin-solidity/contracts/access/Ownable.sol";import "openzeppelin-solidity/contracts/utils/math/SafeMath.sol";import "openzeppelin-solidity/contracts/utils/Strings.sol";import "openzeppelin-solidity/contracts/token/ERC721/extensions/ERC721URIStorage.sol";import "./common/meta-transactions/ContentMixin.sol";import "./common/meta-transactions/NativeMetaTransaction.sol";contract OwnableDelegateProxy {}contract ProxyRegistry { mapping(address => OwnableDelegateProxy) public proxies;}/\*\* \* @title ERC721Tradable \* ERC721Tradable - ERC721 contract that whitelists a trading address, and has minting functionality. \*/abstract contract ERC721Tradable is ContextMixin, ERC721Enumerable, NativeMetaTransaction, Ownable, ERC721URIStorage { using SafeMath for uint256; address proxyRegistryAddress; uint256 private \_currentTokenId = 0; constructor( string memory \_name, string memory \_symbol, address \_proxyRegistryAddress ) ERC721(\_name, \_symbol) { proxyRegistryAddress = \_proxyRegistryAddress; \_initializeEIP712(\_name); } /\*\* \* @dev Mints a token to an address with a tokenURI. \* @param \_to address of the future owner of the token \*/ function mintTo(address \_to, string memory metadataURI) public onlyOwner { uint256 newTokenId = \_getNextTokenId(); \_mint(\_to, newTokenId); \_setTokenURI(newTokenId, metadataURI); \_incrementTokenId(); } /\*\* \* @dev calculates the next token ID based on value of \_currentTokenId \* @return uint256 for the next token ID \*/ function \_getNextTokenId() private view returns (uint256) { return \_currentTokenId.add(1); } /\*\* \* @dev increments the value of \_currentTokenId \*/ function \_incrementTokenId() private { \_currentTokenId++; } function baseTokenURI() virtual public pure returns (string memory); function \_burn(uint256 tokenId) internal override(ERC721, ERC721URIStorage) { super.\_burn(tokenId); } function tokenURI(uint256 tokenId) public view override(ERC721, ERC721URIStorage) returns (string memory) { return super.tokenURI(tokenId); } 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); } /\*\* \* Override isApprovedForAll to whitelist user's OpenSea proxy accounts to enable gas-less listings. \*/ function isApprovedForAll(address owner, address operator) override public view returns (bool) { // Whitelist OpenSea proxy contract for easy trading. ProxyRegistry proxyRegistry = ProxyRegistry(proxyRegistryAddress); if (address(proxyRegistry.proxies(owner)) == operator) { return true; } return super.isApprovedForAll(owner, operator); } /\*\* \* This is used instead of msg.sender as transactions won't be sent by the original token owner, but by OpenSea. \*/ function \_msgSender() internal override view returns (address sender) { return ContextMixin.msgSender(); }}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
and Creature.sol:  
  
// SPDX-License-Identifier: MITpragma solidity ^0.8.0;import "./ERC721Tradable.sol";/\*\* \* @title Creature \* Creature - a contract for my non-fungible creatures. \*/contract Creature is ERC721Tradable{ constructor(address \_proxyRegistryAddress) ERC721Tradable("ExampleNFT", "EXA", \_proxyRegistryAddress) {} function baseTokenURI() override public pure returns (string memory) { return "https://example.com/api/token/"; } function contractURI() public pure returns (string memory) { return "https://creatures-api.opensea.io/contract/opensea-creatures"; } }  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**To deploy: first deploy Creature.sol, second CreatureFactory, and the transferOwnership from Creature to CreatureFactory address

**C2:**Did you ever get this to work? I'm really struggling with it, tried the above steps to no avail, what proxy address should I use? I'm trying with the OpenSea listed Wyvern one.

0 **Answer**