Introduction to Ethereum and Smart Contracts

ddas.tech/introduction-to-ethereum-and-smart-contracts/

April 9, 2023

Created By: Debasis Das (Apr 2023)

In this post we will provide a brief introduction to Ethereum and Smart Contracts such as ERC20 and ERC721.

Table of Contents

What is Ethereum?

- Ethereum is a decentralized, open-source blockchain platform that allows developers to create and deploy smart contracts.
- **Smart contracts** are self-executing contracts with the terms of the agreement between buyer and seller being directly written into lines of code.
- Ethereum-based smart contracts are written in **Solidity**, a programming language designed specifically for Ethereum. These contracts are stored on the Ethereum blockchain, which is a **distributed ledger** that maintains a record of all transactions and smart contracts on the network.
- Ethereum Smart Contracts are transparent and immutable
- Once a smart contract is deployed it cannot be altered providing a high level of security and trust.

Different Types of Smart Contracts

ERC20

- ERC20 is a technical standard for creating tokens on the Ethereum blockchain.
- ERC stands for “Ethereum Request for Comment,” which is a process used to propose and discuss improvements to the Ethereum network.

- ERC20 tokens are created using smart contracts on the Ethereum blockchain and are **fungible**, meaning that each token is interchangeable with every other token of the same type. They can be used for a variety of purposes, such as creating new cryptocurrencies, digital assets, or utility tokens.
- The ERC20 standard defines six mandatory functions and three optional functions that a token smart contract must implement in order to be considered ERC20 compliant.
- These functions include:
    - **totalSupply()** – returns the total supply of the token.
    - **balanceOf(address _owner)** – returns the token balance of a specific address.
    - **transfer(address _to, uint256 _value)** – transfers a specified amount of tokens from the sender's address to another address.
    - **approve(address _spender, uint256 _value)** – allows a third party to spend tokens on behalf of the token holder.
    - **transferFrom(address _from, address _to, uint256 _value)** – allows a third party to transfer tokens from one address to another.
    - **allowance(address _owner, address _spender)** – returns the amount of tokens that a third party is allowed to spend on behalf of the token holder.

ERC721

- ERC721 is a technical standard for creating **non-fungible tokens (NFTs)** on the Ethereum blockchain.
- ERC721 tokens are **unique and non-interchangeable**, meaning that each token has its own distinct properties and cannot be exchanged for another token.
- They can be used for a variety of purposes, such as creating unique digital assets, collectibles, and gaming items.
- The ERC721 standard defines a set of functions that a token smart contract must implement in order to be considered ERC721 compliant. These functions include:
    - **totalSupply()** – returns the total supply of the token.
    - **balanceOf(address _owner)** – returns the token balance of a specific address.
    - **ownerOf(uint256 _tokenId)** – returns the owner of a specific token.
    - **approve(address _approved, uint256 _tokenId)** – allows a third party to transfer ownership of a specific token.
    - **transferFrom(address _from, address _to, uint256 _tokenId)** – transfers ownership of a specific token from one address to another.
    - **tokenMetadata(uint256 _tokenId)** – returns the metadata associated with a specific token.

Differences between ERC20 and ERC721

The main **difference between ERC20 and ERC721 i**s that ERC20 is a fungible token standard, meaning that all tokens are the same and can be exchanged for each other, while ERC721 is a non-fungible token standard, meaning that each token is unique and cannot be

exchanged for another token on a one-to-one basis.

Here are some specific differences between ERC20 and ERC721:

1. **Fungibility**: ERC20 tokens are interchangeable and have identical values, while ERC721 tokens are unique and non-interchangeable.
2. **Supply**: ERC20 tokens have a fixed or variable total supply, while ERC721 tokens can have an unlimited supply of unique, non-interchangeable tokens.
3. **Use cases:** ERC20 tokens are commonly used for representing currencies, assets, or utility tokens, while ERC721 tokens are commonly used for creating unique, one-of-a-kind digital assets, collectibles, and gaming items.
4. **Implementation**: ERC20 tokens require only a few basic functions to be implemented, while ERC721 tokens require more complex functions for managing unique, non-interchangeable tokens.

ERC20 Sample Contract

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MyToken {
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    uint256 public totalSupply;
    mapping(address => uint256) public balanceOf;
    mapping(address => mapping(address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    constructor(uint256 _initialSupply) {
        name = "My Token";
        symbol = "MYT";
        totalSupply = _initialSupply * 10 ** uint256(decimals);
        balanceOf[msg.sender] = totalSupply;
    }

    function transfer(address _to, uint256 _value) public returns (bool success) {
        require(_to != address(0));
        require(balanceOf[msg.sender] >= _value);

        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;

        emit Transfer(msg.sender, _to, _value);
        return true;
    }

    function approve(address _spender, uint256 _value) public returns (bool success)
{
        require(_spender != address(0));

        allowance[msg.sender][_spender] = _value;

        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    function transferFrom(address _from, address _to, uint256 _value) public returns
(bool success) {
        require(_to != address(0));
        require(balanceOf[_from] >= _value);
        require(allowance[_from][msg.sender] >= _value);

        balanceOf[_from] -= _value;
        balanceOf[_to] += _value;
        allowance[_from][msg.sender] -= _value;
```

```
        emit Transfer(_from, _to, _value);
        return true;
    }
}
```

MYTOKEN AT 0XD91...39138 (ME

Balance: 0 ETH

approve address _spender, uint256 _val

transfer address _to, uint256 _value

transferFrom address _from, address _to, uint

allowance address , address

balanceOf balanceOf - call Cfcl

0: uint256: 10000000000000000000

decimals

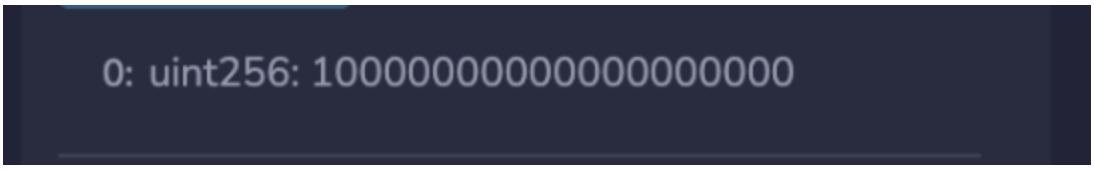
name

0: string: My Token

symbol

0: string: MYT

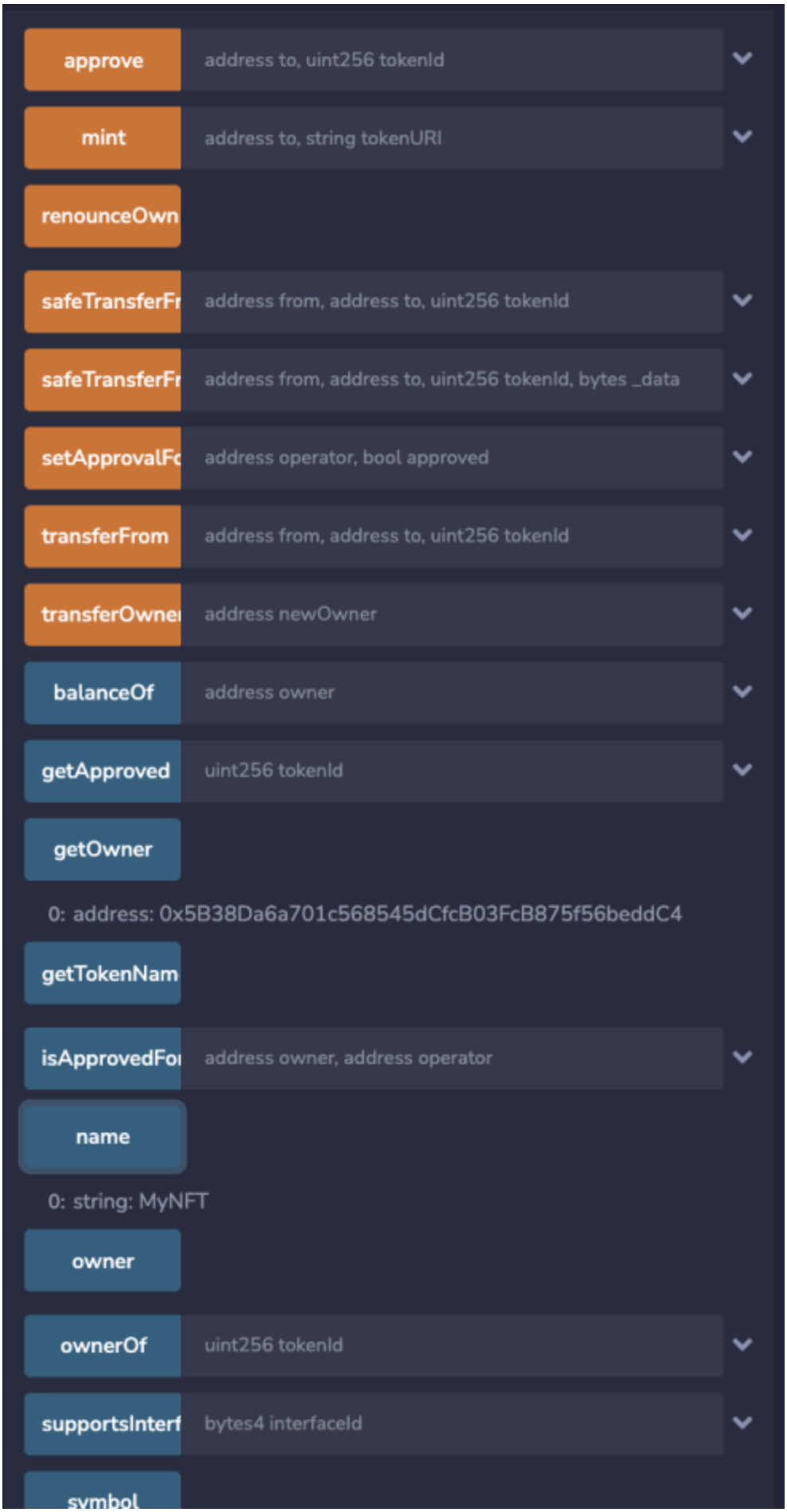
totalSupply

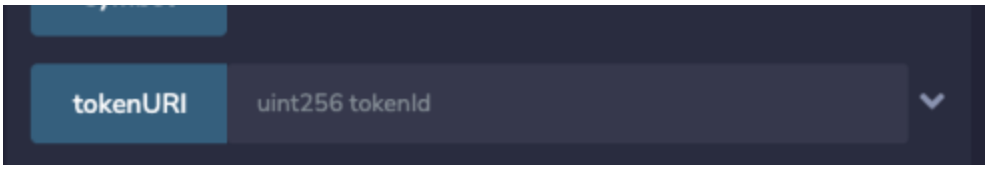


ERC 20 Sample Contract – Deployment and Outputs

ERC721 Sample Contract

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.7;

import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v4.1.0/contracts/token/ERC721/ERC721.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v4.1.0/contracts/utils/Counters.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v4.1.0/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/v4.1.0/contracts/access/Ownable.sol";

contract MyNFT is ERC721URIStorage,Ownable {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;
    event MyEvent(uint256 message);

    constructor() ERC721("MyNFT", "NFT") {}

    function getTokenName() public view returns (string memory) {
        return name();
    }

    function getOwner() public view returns (address) {
        return owner();
    }

    function mint(address to, string memory tokenURI) public returns (uint256) {
        require(msg.sender == owner(), "Only the contract owner can mint tokens");
        _tokenIds.increment();
        uint256 newTokenId = _tokenIds.current();
        _mint(to, newTokenId);
        _setTokenURI(newTokenId, tokenURI);
        return newTokenId;
    }
}
```


approve
address to, uint256 tokenId
mint
address to, string tokenURI
renounceOwn
safeTransferFr
address from, address to, uint256 tokenId
safeTransferFr
address from, address to, uint256 tokenId, bytes _data
setApprovalFo
address operator, bool approved
transferFrom
address from, address to, uint256 tokenId
transferOwne
address newOwner
balanceOf
address owner
getApproved
uint256 tokenId
getOwner
0: address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
getTokenNam
isApprovedFo
address owner, address operator
name
0: string: MyNFT
owner
ownerOf
uint256 tokenId
supportsInterf
bytes4 interfaceId
symbol



ERC721 Contract Deployment & Outputs