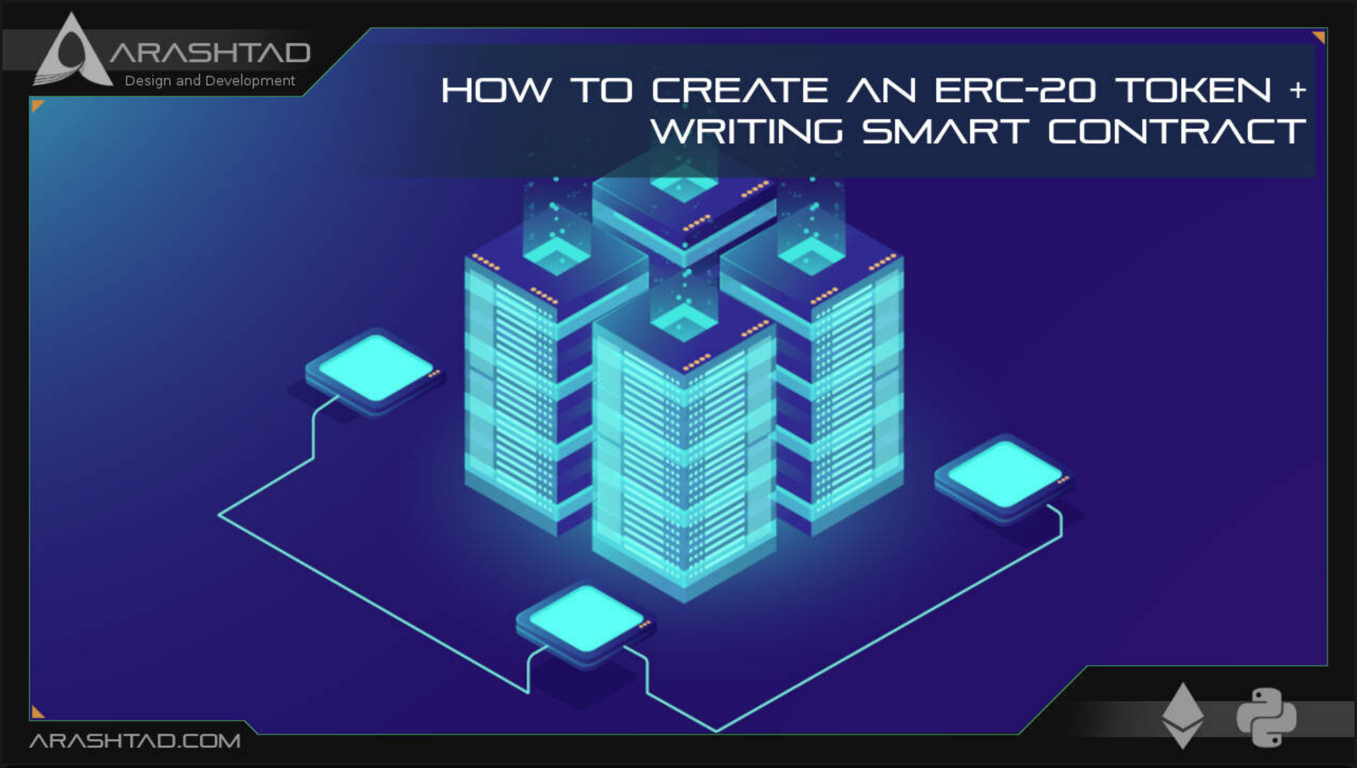
How to Create an ERC-20 Token + Writing Smart Contract

No comments



In this tutorial, we are going to get familiar with the methods and events of the ERC-20 token. In addition to that, we want to create an ERC-20 token on the Rinkeby Testnet. By writing the smart contract, the deployment python file, helpful_scripts.py, and other files that help the creation and deployment of the ERC-20 token smart contract. This tutorial will go on in the next articles where we will explain the ERC-20 smart contract deeper and provide guidelines to add the created tokens to your Metamask wallet.

What Is ERC-20?

The ERC-20 is a token standard that implements an API for tokens within smart contracts. The acronym stands for Ethereum Request for Comments 20. The ERC-20 proposes a standard for fungible tokens (as opposed to Non-Fungible Tokens or NFTs that use ERC- 721) meaning that they have a property that makes each of them the same. This standard provides the following functionalities:

1. Transfering tokens from one account to another.

2. Tetting the balance of an account.

3. Getting the total supply of the token available on the network.

4. Approving whether a third party can spend tokens from an account.

Methods and Events of an ERC-20 Token

If a smart contract implements the events and methods of ERC-20, it can be called an ERC-20 token contract.

Methods:

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns
 (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool
 success)
function transferFrom(address _from, address _to, uint256 _value)
public returns (bool success)
function approve(address _spender, uint256 _value) public returns (
bool success)
function allowance(address _owner, address _spender) public view
returns (uint256 remaining)
```

Events:

```
event Transfer(address indexed _from, address indexed _to, uint256
_value)
event Approval(address indexed _owner, address indexed _spender,
uint256 _value)
```

Getting Started with the ERC-20 Token in Brownie

Now, we begin our project by typing in the terminal:

```
brownie init
```

And in our contracts folder, we create a file named OurToken.sol and paste the below contract we have copied from this link. You can change the name of the token whatever you want:

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

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract OurToken is ERC20 {
// wei
 constructor(uint256 initialSupply)ERC20("OurToken", "OT") {
_mint(msg.sender, initialSupply);
 }
}
```

We also create a deploy_token.py file in the scripts folder and paste the below code in it:

```
from brownie import accounts, config, TokenERC20, EasyToken
from scripts.helpful_scripts import get_account

initial_supply = 1000000000000000000000 # 1000
token_name = "Token"
token_symbol = "TKN"

def main():
account = get_account()
erc20 = TokenERC20.deploy(initial_supply, token_name, token_symbol, {
"from": account})
```

And helpful_scripts.py goes like this:

```
from brownie import network, accounts, config

LOCAL_BLOCKCHAIN_ENVIRONMENTS = ["hardhat", "development",
"mainnet-fork"]

def get_account():
if network.show_active() in LOCAL_BLOCKCHAIN_ENVIRONMENTS:
return accounts[0]
if network.show_active() in config["networks"]:
  account= accounts.add(config["wallets"]["from_key"])
return account
return None
```

ERC-20 Token Smart Contract

And finally this is the contract that contains the ERC-20 methods and events (TokenERC20.sol):

```
pragma solidity ^0.6.0;

interface tokenRecipient {
 function receiveApproval(address_from, uint256 _value, address _token
, bytes  calldata_extraData) external;
 }

contract TokenERC20 {
// Public variables of the token
 string publicname;
 string public symbol;
 uint8 public decimals =18;
// 18 decimals is the strongly suggested default, avoid changing it
 uint256 public totalSupply;
// This creates an array with all balances
 mapping (address => uint256) public balanceOf;
 mapping (address => mapping (address => uint256)) public allowance;
// This generates a public event on the blockchain that will notify clients

 event Transfer(address indexedfrom, address indexed to
, uint256 value);
// This generates a public event on the blockchain that will notify clients

 event Approval(address indexed_owner, address indexed _spender
, uint256 _value);
 event Burn(address indexedfrom, uint256 value);
 constructor(
   uint256 initialSupply,
   string memory tokenName,
   string memory tokenSymbol
   ) public {
  totalSupply = initialSupply *10 ** uint256(decimals);
                //Update total supply with the decimal amount

  balanceOf[msg.sender] = totalSupply;
// Give the creator all initial tokens
name = tokenName; // Set the name for display purposes
  symbol = tokenSymbol;// Set the symbol for display purposes
 }

 function_transfer(address _from, address _to, uint _value) internal {
// Prevent transfer to 0x0 address. Use burn() instead
  require(_to != address(0x0));
// Check if the sender has enough
  require(balanceOf[_from] >= _value);
```

```
// Check for overflows
  require(balanceOf[_to] + _value >= balanceOf[_to]);
// Save this for an assertion in the future
  uint previousBalances = balanceOf[_from] + balanceOf[_to];
// Subtract from the sender
  balanceOf[_from] -= _value;
// Add the same to the recipient
  balanceOf[_to] += _value;
  emit Transfer(_from, _to, _value);
// Asserts are used to use static analysis to find bugs in your code. They sh

                //fail
assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
 }


 function transfer(address _to, uint256 _value
) public returns (bool success) {
_transfer(msg.sender, _to, _value);
  return true;
 }

 function transferFrom(address _from, address _to, uint256 _value
) public returns (bool success) {
  require(_value <= allowance[_from][msg.sender]); // Check allowance
  allowance[_from][msg.sender] -= _value;
_transfer(_from, _to, _value);
  return true;
 }

 function approve(address _spender, uint256 _value
) public returns (bool success) {
  allowance[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
 }

 function approveAndCall(address _spender, uint256 _value
, bytes memory _extraData)
                             public returns (bool success) {
  tokenRecipient spender = tokenRecipient(_spender);
if (approve(_spender, _value)) {
                             spender.receiveApproval(msg.sender,
_value, address(this), _extraData);
   return true;
  }
 }

 function burn(uint256 _value) public returns (bool success) {
  require(balanceOf[msg.sender] >= _value);
```

```
// Check if the sender has enough
  balanceOf[msg.sender] -=value; // Subtract from the sender
  totalSupply -=value; // Updates totalSupply
  emit Burn(msg.sender, value);
  returntrue;
 }

 function burnFrom(address_from, uint256 _value
) public returns (bool success) {
  require(balanceOf[from] >= _value);
// Check if the targeted balance is enough
  require(value <= allowance[_from][msg.sender]); // Check allowance
  balanceOf[from] -= _value; // Subtract from the targeted balance
  allowance[from][msg.sender] -= _value;
// Subtract from the sender's allowance
  totalSupply -=value; // Update totalSupply
  emit Burn(from, _value);
  returntrue;
 }
}
```

We are going to explain the above contract deeper in the next article. But for now, it is time to deploy the contract itself:

```
brownie run scripts/1_deploy_token.py
```

Result:

```
Brownie v1.18.1 - Python development framework for
EthereumErc20BrownieProject is the active project.Launching 'ganache-
cli --chain.vmErrorsOnRPCResponse true --wallet.totalAccounts 10 --
hardfork istanbul --miner.blockGasLimit 12000000 --wallet.mnemonic
brownie --server.port 8545'...Running
'scripts/1_deploy_token.py::main'... Transaction sent:
0xe677ec7a8e17a7e750bb65f908bf77c92b5bbd368d975f7c3b36131544c9cf02 Gas
price: 0.0 gwei Gas limit: 12000000 Nonce: 0 TokenERC20.constructor
confirmed Block: 1 Gas used: 670648 (5.59%) TokenERC20 deployed at:
0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87Terminating local RPC
client…
```

From the above results, we are going to use the address that the smart contract is deployed at the "TokenERC20" Which is 0x3194cBDC3dbcd3E11a07892e7bA5c3394048Cc87 in this case, and track in the Etherscan.

Final Word

In this article, we have got familiar with the methods and events of an ERC-20 token. Besides, we have created a project containing the ERC-20 token smart contract, the deployment python files, and so on to create a typical ERC-20 token with a certain number of tokens created in our Metamask wallet. In the tutorials, we will see how we can add these tokens in the Uniswap and also find them on the Etherscan.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.

Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP NEWSLETTER RSS FEED