

What is an ERC?

What is an EIP?

18.59.28

<https://eips.ethereum.org/>

EIP → Ethereum Improvement Proposals

<http://github.com/ethereum/EIPs>

ALL BLOCKCHAINS HAVE THEIR
STANDARDS LIKE eip or erc

ERC-20 TOKEN STANDARD

<https://ethereum.org/en/developers/docs/standards/tokens/erc-20/>

ERC → Ethereum Request for Comments

What is an ERC20?

19.01.40

What are ERC20?



- tokens are deployed on a chain using ERC20 token standard

<https://eips.ethereum.org/EIPS/eip-20>

- basically its a smart contract that actually represents a token.
- technically chain link is the ERC 677 as there are upgrades to the ERC20 that some tokens take that are still backwards compatible ERC20s
- ERC 677 and ERC 777 are compatible with ERC20

Why make an ERC20?

- 1. Governance Tokens**
- 2. Secure an underlying network**
- 3. Create a synthetic asset**
- 4. Or anything else**

Manually Creating an ERC20 Token 19.03.06

```
Mkdir hardhat-erc20-fcc  
Cd hardhat-erc20-fcc  
yarn add --dev hardhat
```

```
code .
```

```
Yarn hardhat  
Create an empty hardhat.config.js
```

```
Add hardhat.config.js from the old Project and .env settings
```

```

require("@nomiclabs/hardhat-waffle")
require("@nomiclabs/hardhat-etherscan")
require("hardhat-deploy")
require("solidity-coverage")
require("hardhat-gas-reporter")
require("hardhat-contract-sizer")
require("dotenv").config()

/**
 * @type
import('hardhat/config').HardhatUserConfig
 */

const RINKEBY_RPC_URL =
process.env.RINKEBY_RPC_URL
const PRIVATE_KEY = process.env.PRIVATE_KEY
const ETHERSCAN_API_KEY =
process.env.ETHERSCAN_API_KEY
const COINMARKETCAP_API_KEY =
process.env.COINMARKETCAP_API_KEY

module.exports = {
  solidity: "0.8.8",
  networks: {

```

```

hardhat: {
    chainId: 31337,
    blockConfirmations: 1,
  },
  localhost: {
    chainId: 31337,
  },
  rinkeby: {
    chainId: 4,
    blockConfirmations: 6,
    url: RINKEBY_RPC_URL,
    accounts: [PRIVATE_KEY],
  },
},
namedAccounts: {
  deployer: {
    default: 0,
  },
  player: {
    default: 1,
  },
},
},

```



```
gasReporter: {
  enabled: true,
  outputFile: "gas-report.txt",
  noColors: true,
  currency: "USD",
  coinmarketcap: COINMARKETCAP_API_KEY,
  token: "ETH",
},
mocha: {
  timeout: 500000, // 200 seconds max
},
etherscan: {
  apiKey: ETHERSCAN_API_KEY,
},
}
```


Add folder and file

contracts/ManualToken.sol

- The main reason this token (smart contract) Works is that theres some balances mapping.

```
mapping (address => uint256) public balanceOf;
```

address key is going to be every single address on the planet, and then how much they have

- transfer tokens : subtract from address amount and add to the address

```
// this is pseudocode and not a full transfer function  
// there will be some assert, requires and will complete end of the
```

education

```
function _transfer(address from, address to, uint256 amount) public () {  
    balanceOf[from] = balanceOf[from] - amount;  
    balanceOf[to] += amount;  
}
```

Transfer Works when the caller is sending Money directly in to another address.

What happens if we want to allow some smart contract to work with our token, or we want to allow somebody else to work with our token??

- to deposit it into a protocol
- or do some more functionality with it

There will be some approved function that will approve that contract to do that.

```
function tranferFrom()public(){  
    // implement taking funds from a user  
}
```

There will be some type of allowances mapping that will tell whos allowed which address to take how much token,

```
mapping (address =>mapping(address=>uint256)) public allowance;
```

```

    /**
    * Transfer tokens from other address
    *
    * Send `_value` tokens to `_to` on behalf of `_from`
    *
    * @param _from The address of the sender
    * @param _to The address of the recipient
    * @param _value the amount to send
    */
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; // update the allowance
    _transfer(_from, _to, _value); // transfer tokens.
    return true;
}

```

We have to implement how many tokens there are starting with how many tokens there are total ??? Sometimes you will add a limit function to add more functions

```
uint256 initialSupply;
```

COPY ALL CODE FROM GITHUB REPO=====

Creating an ERC20 Token with Openzeppelin

19.09.24

<https://www.openzeppelin.com/>

<https://github.com/OpenZeppelin/openzeppelin-contracts>

<https://docs.openzeppelin.com/contracts/4.x/>

Create ourtoken.sol file

contracts/OurToken.sol

```
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.8;  
  
contract OurToken {}
```

Install openzeppelin contracts

```
yarn add --dev @openzeppelin/contracts
```



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

```
contract OurToken is ERC20 {}
```

```
contract OurToken is ERC20 {  
    constructor() ERC20("OurToken", "OT") {}  
    // OurToken is token name and OT is token symbol  
}
```

To make token with these functions, actually get initialized with zero tokens.

To make some token, we using mint functions

Mint function is to allows us to create tokens,

```
contract OurToken is ERC20 {  
    // initial supply is 50 <- 50 WEI  
    // initial supply 50e18  
    // 50 * 10**18  
    // because of decimal function - it returns 18  
    constructor(uint256 initialSupply) ERC20("OurToken", "OT") {  
        _mint(msg.sender, initialSupply);  
    }  
}
```

son