---

title: Understand the ERC-20 token smart contract description: An introduction to deploying your first smart contract on an Ethereum test network author: "jdourlens" tags: ["smart contracts", "tokens", "solidity", "erc-20"] skill: beginner lang: en published: 2020-04-05 source: EthereumDev sourceUrl: https://ethereumdev.io/understand-the-erc20-token-smart-contract/ address: "0x19dE91Af973F404EDF5B4c093983a7c6E3EC8ccE"

One of the most significant smart contract standards on Ethereum is known as ERC-20, which has emerged as the technical standard used for all smart contracts on the Ethereum blockchain for fungible token implementations.

ERC-20 defines a common list of rules that all fungible Ethereum tokens should adhere to. Consequently, this token standard empowers developers of all types to accurately predict how new tokens will function within the larger Ethereum system. This simplifies and eases developers' tasks, because they can proceed with their work, knowing that each and every new project won't need to be redone every time a new token is released, as long as the token follows the rules.

Here is, presented as an interface, the functions an ERC-20 must implement. If you're not sure about what is an interface: check our article about OOP programming in Solidity.

```solidity pragma solidity ^0.6.0;

interface IERC20 {

```
function totalSupply() external view returns (uint256);
function balanceOf(address account) external view returns (uint256);
function allowance(address owner, address spender) external view returns (uint256);

function transfer(address recipient, uint256 amount) external returns (bool);
function approve(address spender, uint256 amount) external returns (bool);
function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

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

} ```

Here is a line-by-line explainer of what every function is for. After this we'll present a simple implementation of the ERC-20 token.

## Getters {#getters}

`solidity function totalSupply() external view returns (uint256);`

Returns the amount of tokens in existence. This function is a getter and does not modify the state of the contract. Keep in mind that there are no floats in Solidity. Therefore most tokens adopt 18 decimals and will return the total supply and other results as followed 1000000000000000000 for 1 token. Not every token has 18 decimals and this is something you really need to watch for when dealing with tokens.

`solidity function balanceOf(address account) external view returns (uint256);`

Returns the amount of tokens owned by an address (`account`). This function is a getter and does not modify the state of the contract.

`solidity function allowance(address owner, address spender) external view returns (uint256);`

The ERC-20 standard allows an address to give an allowance to another address to be able to retrieve tokens from it. This getter returns the remaining number of tokens that the `spender` will be allowed to spend on behalf of `owner`. This function is a getter and does not modify the state of the contract and should return 0 by default.

## Functions {#functions}

`solidity function transfer(address recipient, uint256 amount) external returns (bool);`

Moves the `amount` of tokens from the function caller address (`msg.sender`) to the recipient address. This function emits the
```

`Transfer` event defined later. It returns true if the transfer was possible.

```
solidity function approve(address spender, uint256 amount) external returns (bool);
```

Set the amount of `allowance` the `spender` is allowed to transfer from the function caller (`msg.sender`) balance. This function emits the Approval event. The function returns whether the allowance was successfully set.

```
solidity function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
```

Moves the `amount` of tokens from `sender` to `recipient` using the allowance mechanism. amount is then deducted from the caller's allowance. This function emits the `Transfer` event.

Events {#events}

```
solidity event Transfer(address indexed from, address indexed to, uint256 value);
```

This event is emitted when the amount of tokens (value) is sent from the `from` address to the `to` address.

In the case of minting new tokens, the transfer is usually `from` the 0x00..0000 address while in the case of burning tokens the transfer is `to` 0x00..0000.

```
solidity event Approval(address indexed owner, address indexed spender, uint256 value);
```

This event is emitted when the amount of tokens (`value`) is approved by the `owner` to be used by the `spender`.

A basic implementation of ERC-20 tokens {#a-basic-implementation-of-erc-20-tokens}

Here is the most simple code to base your ERC-20 token from:

```solidity pragma solidity ^0.8.0;

interface IERC20 {

```
function totalSupply() external view returns (uint256);
function balanceOf(address account) external view returns (uint256);
function allowance(address owner, address spender) external view returns (uint256);

function transfer(address recipient, uint256 amount) external returns (bool);
function approve(address spender, uint256 amount) external returns (bool);
function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

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

}

contract ERC20Basic is IERC20 {

```
string public constant name = "ERC20Basic";
string public constant symbol = "ERC";
uint8 public constant decimals = 18;

mapping(address => uint256) balances;

mapping(address => mapping (address => uint256)) allowed;

uint256 totalSupply_ = 10 ether;
```

constructor() { balances[msg.sender] = totalSupply_; }

```
function totalSupply() public override view returns (uint256) {
return totalSupply_;
}

function balanceOf(address tokenOwner) public override view returns (uint256) {
    return balances[tokenOwner];
}
```
```

```
function transfer(address receiver, uint256 numTokens) public override returns (bool) {
    require(numTokens <= balances[msg.sender]);
    balances[msg.sender] = balances[msg.sender]-numTokens;
    balances[receiver] = balances[receiver]+numTokens;
    emit Transfer(msg.sender, receiver, numTokens);
    return true;
}

function approve(address delegate, uint256 numTokens) public override returns (bool) {
    allowed[msg.sender][delegate] = numTokens;
    emit Approval(msg.sender, delegate, numTokens);
    return true;
}

function allowance(address owner, address delegate) public override view returns (uint) {
    return allowed[owner][delegate];
}

function transferFrom(address owner, address buyer, uint256 numTokens) public override returns (bool) {
    require(numTokens <= balances[owner]);
    require(numTokens <= allowed[owner][msg.sender]);

    balances[owner] = balances[owner]-numTokens;
    allowed[owner][msg.sender] = allowed[owner][msg.sender]-numTokens;
    balances[buyer] = balances[buyer]+numTokens;
    emit Transfer(owner, buyer, numTokens);
    return true;
}
```

} ```

Another excellent implementation of the ERC-20 token standard is theOpenZeppelin ERC-20 implementation.