Steven McKie   Follow

Writer. Programmer. UX/UI. Bitcoin/Ethereum. ENTP. Doing things
Sep 30, 2017 · 4 min read

The Anatomy of ERC20

What's on the Inside of Ethereum's Most
Popular Contract

Crosspost: This was originally posted by Gerald Nash ⚡ , here. It was reposted to BlockChannel with their permission.

In light of today's ICO launches, digital token sales during which companies and organizations raise millions of dollars instantly while giving out digital assets, it's important to recognize the underlying technology that nearly all of these tokens possess: ERC20.

Ethereum Request for Comments 20, or ERC20, is an Ethereum Improvement Proposal introduced by Fabian Vogelsteller in late 2015. It's a standard by which many popular Ethereum smart contracts abide. It effectively allows smart contracts to act very similarly to a conventional cryptocurrency like Bitcoin, or Ethereum itself. In saying this, a token hosted on the Ethereum blockchain can be sent, received, checked of its total supply, and checked for the amount that is available

on an individual address. This is analogous to sending and receiving
Ether or Bitcoin from a wallet, knowing the total amount of coins in
circulation, and knowing a particular wallet's balance of a coin. A smart
contract that follows this standard is called an ERC20 token.

ERC: Token standard · Issue #20 · ethereum/EIPs

ERC: 20 Title: Token standard Status:
Draft Type: Informational Created:
19–11.2015 Resolution:
https://github.com...

github.com
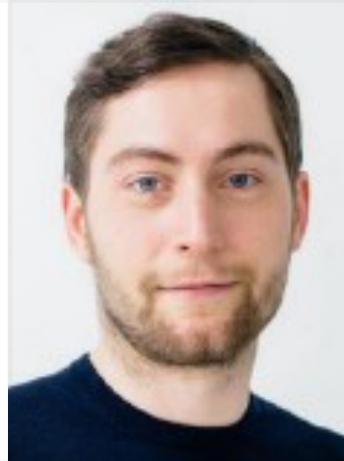
All of the previously described functionality is able to exist by defining a
set of functions that allow a smart contract to emulate a digital token.
But how does that work?

ERC20 defines the functions `balanceOf` , `totalSupply` , `transfer` ,
`transferFrom` , `approve` , and `allowance` . It also has a few optional
fields like the token name, symbol, and the number of decimal places
with which it will be measured.

Note: This is a concise declaration of an example ERC20 contract.

```
1    // Grabbed from: https://github.com/ethereum/EIPs/issues/20
2    contract ERC20 {
3        function totalSupply() constant returns (uint theTotalSu
4        function balanceOf(address _owner) constant returns (uin
5        function transfer(address _to, uint _value) returns (boo
6        function transferFrom(address _from, address _to, uint _
7        function approve(address _spender, uint _value) returns
8        function allowance(address _owner, address _spender) con
```

An overview and example of each field within the contract is as follows.

totalSupply()

Although the supply could easily be fixed, as it is with Bitcoin, this function allows an instance of the contract to calculate and return the total amount of the token that exists in circulation.

```
1   contract MyERCToken {
2       // In this case, the total supply
3       // of MyERCToken is fixed, but
4       // it can very much be changed
5       uint256 _totalSupply = 1000000;
6
7       function totalSupply() constant returns (uint256 theTotal
8           // Because our function signature
9           // states that the returning variable
10          // is "theTotalSupply", we'll just set that variable
11          // to the value of the instance variable "_totalSupply"
```

balanceOf()

This function allows a smart contract to store and return the balance of the provided address. The function accepts an address as a parameter, so it should be known that the balance of any address is public.

```
1   contract MyERCToken {
2       // Create a table so that we can map addresses
3       // to the balances associated with them
4       mapping(address => uint256) balances;
5       // Owner of this contract
6       address public owner;
7
8       function balanceOf(address _owner) constant returns (uint
9           // Return the balance for the specific address
```

approve()

When calling this function, the owner of the contract authorizes, or *approves*, the given address to withdraw instances of the token from the owner's address.

Here, and in later snippets, you may see a variable `msg` . This is an implicit field provided by external applications such as wallets so that they can better interact with the contract. The Ethereum Virtual Machine (EVM) lets us use this field to store and process data given by the external application.

In this example, `msg.sender` is the address of the contract owner.

```
 1    contract MyERCToken {
 2       // Create a table so that we can map
 3       // the addresses of contract owners to
 4       // those who are allowed to utilize the owner's contract
 5       mapping(address => mapping (address => uint256)) allowed;
 6
 7       function approve(address _spender, uint256 _amount) retur
 8          allowed[msg.sender][_spender] = _amount;
 9          // Fire the event "Approval" to execute any logic
10          // that was listening to it
```

transfer()

This function lets the owner of the contract send a given amount of the token to another address just like a conventional cryptocurrency transaction.

```
1   contract MyERCToken {
2     mapping(address => uint256) balances;
3
4     // Note: This function returns a boolean value
5     //       indicating whether the transfer was successful
6     function transfer(address _to, uint256 _amount) returns (
7       // If the sender has sufficient funds to send
8       // and the amount is not zero, then send to
9       // the given address
10      if (balances[msg.sender] >= _amount
11        && _amount > 0
12        && balances[_to] + _amount > balances[_to]) {
13        balances[msg.sender] -= _amount;
14        balances[_to] += _amount;
15        // Fire a transfer event for any
```

transferFrom()

This function allows a smart contract to automate the transfer process and send a given amount of the token on behalf of the owner.

Seeing this might raise a few eyebrows. One may question why we need both `transfer()` and `transferFrom()` functions.

Consider transferring money to pay a bill. It's extremely common to send money manually by taking the time to write a check and mail it to pay the bill off. This is like using `transfer()` : you're doing the money transfer process yourself, without the help of another party.

In another situation, you could set up automatic bill pay with your bank. This is like using `transferFrom()` : your bank's machines send money to pay off the bill on your behalf, automatically. With this function, a contract can send a certain amount of the token to another address on your behalf, without your intervention.

```
1    contract MyERCToken {
2      mapping(address => uint256) balances;
3
4      function transferFrom(address _from, address _to, uint256
5        if (balances[_from] >= _amount
6          && allowed[_from][msg.sender] >= _amount
7          && _amount > 0
8          && balances[_to] + _amount > balances[_to]) {
9        balances[_from] -= _amount;
10       balances[_to] += _amount;
11       Transfer(_from, _to, _amount);
12         return true;
```

Token Name

This is an optional field, but many popular tokens include it so that popular wallets like Mist and MyEtherWallet are able to identify them.

```
contract MyERCToken {
  string public constant name = "My Custom ERC20 Token";


}
```

Token Symbol

Another optional field used to identify a token, this is a three or four letter abbreviation of the token, just like BTC, ETH, AUG, or SJCX.

```
contract MyERCToken {
  string public constant symbol = "MET";


}
```

Number of Decimals

An optional field used to determine to what decimal place the amount of the token will be calculated. The most common number of decimals to consider is 18.

```
contract MyERCToken {
  uint8 public constant decimals = 18;


}
```

The total source code of ERC20 token that we just created can be found here.

In the end, the original ERC20 proposal is rather unappreciated. It opened up avenues for a new set of smart contracts that could be created and distributed in the same fashion as Bitcoin or Ethereum. This proves to be very enticing for young companies, as the entire ERC20 ecosystem is hosted on the Ethereum blockchain, a large pre-existing network of computers. This means that developers and young companies don't have to attract miners in order to sustain their tokens, something that can save a lot of money. And, these tokens can be hosted on exchanges to be traded like other assets, so investors can easily buy and sell these tokens like more popular currencies.