

Forum:[Hack Forums](#)

Thread: [ERC20 Currency Token - Solidity Smart Contract Example](#)

Hackers - May 16, 2024, 4:54 am

Fallen Currency Token Example

Hi all, today you will learn how to create a smart contract that allows you to have your own custom ERC20 token. This can be used as a currency so you can issue tokens for any reason. The token itself does not have any functionality on-contract other than the default OpenZeppelin ERC20 features.

Let me break down the whole contract for you, but if you want to copy the full source code you can do so - it's in the spoiler at end of my post!

Code // SPDX-License-Identifier: MIT

pragma solidity ^0.8.0; This is our licensing identifier (MIT) and the Solidity version we are using. Many Solidity versions are similar enough, and I can verify that other versions will work for the code below.

Code contract FallenExampleToken is ERC20, Ownable { When you create a smart contract, you define the contract name like this in your code, which is how web3 related tools will target it and deploy it. I recommend checking out <https://remix.ethereum.org> so you do not need to worry about the technical requirements for deploying and can do it straight from there with a test wallet on testnet, or a web3 wallet extension for mainnet.

Code uint256 public constant MAX_SUPPLY = 1699786 * 10**18; // 18 max decimal places, max supply of 1699786 'coins' The max supply is set here. In this case, I chose [Charlie Sheen](#) 's UID (1699786) as the max amount of "coins" / "tokens", but with 18 max decimal places. What this means is that the tracking occurs with the "sub units" (down to the 18th decimal place) for the main coins. This is what allows fractional token transfers instead of forcing transfers with only 1+ (integers).

```
Code constructor() ERC20("FallenExampleToken", "FET") {  
    _mint(msg.sender, MAX_SUPPLY);  
}
```

The constructor runs once when the contract is deployed. In this case, we are giving our coin a full name & a nickname. This is what will show our token abbreviation on blockchain scanners and data.

In this case, we are minting the total max supply to the address that deploys the contract. You can then use transfer functions to send a balance from your owner address to another user. In other contracts you could remove this _mint() func use in the constructor, which would leave all tokens unminted. This would allow you to then mint on demand later on instead of having them pre-minted. Of course, the amount of logic & special rules you add is up to you.

```
Code function mint(address to, uint256 amount) external onlyOwner {
    require(totalSupply() + amount <= MAX_SUPPLY, "Exceeds maximum token supply");
    _mint(to, amount);
}
```

This is the individual mint function above that is used. It checks to make sure the new amount being minted will not cause the total supply to exceed the max. If so, it will reject it with the require statement's error. If it can proceed, it will mint the new tokens to the address.

I am not showing the transfer function because it's a core part of OpenZeppelin's ERC20 contracts, so you can refer to their documentation on those types of processes.

```
Code function decimals() public view virtual override returns (uint8) {
    return 18;
}
```

This is what allows external systems to see that this FallenExampleToken (FET) is tracking up to 18 decimal places of sub-tokens, but this is what lets people know what the main token is & how many subtokens make up 1 token.

Here is the full source code for you to use. You can deploy this as-is and have your own currency immediately. Enjoy! :)

Spoiler ([Click to View](#))

Code // SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

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

import "@openzeppelin/contracts/access/Ownable.sol";

```
contract FallenExampleToken is ERC20, Ownable {
    uint256 public constant MAX_SUPPLY = 1699786 * 10**18; // 18 max decimal places, max
supply of 1699786 'coins'
```

```
    constructor() ERC20("FallenExampleToken", "FET") {
        _mint(msg.sender, MAX_SUPPLY);
    }
```

```
    function mint(address to, uint256 amount) external onlyOwner {
        require(totalSupply() + amount <= MAX_SUPPLY, "Exceeds maximum token supply");
        _mint(to, amount);
    }
```

```
    function decimals() public view virtual override returns (uint8) {
        return 18;
    }
}
```

Looking for more? Check this out:

<https://hackforums.net/showthread.php?tid=6271201>

Euaggelistes - May 16, 2024, 4:59 am

Very nice. Thank you for sharing. It's a good introduction to Solidity.

Hackers - May 16, 2024, 5:30 am

(May 16, 2024 04:59 AM)Euaggelistes Wrote: [\[CITE\]](#) Very nice. Thank you for sharing. It's a good introduction to Solidity.

No problem & thanks!

It's not a hard language to pick up, but can be very tedious at first. I like it a lot overall. Been a while since I made a new contract so figured I'd share some examples on HF.

If anyone needs any help with this or anything just reply here guys :)

Stanley - May 16, 2024, 4:42 pm

I'm Stanley HF's helper bot. The OP (original poster) of this thread wants everyone to see this page and has elected to bump it to the top of the showthread listing. Since HF doesn't want a ton of LQP (Low Quality Posts) this is the best method of bumping your content. There is a small ?yte fee for each bump. Any OP can bump any of their threads on HF from the linked icon on the top of their first post.

If you haven't already read the first post of this thread we encourage you to do so.

Thanks,
Stanley