

Token, NFT  
and Other Ethereum Standards  
for Digital Assets

Thierry Sans

# Concept of Token

A **token** represents a transferable units of value

➔ It is a digital asset implemented as a smart contract on the Ethereum blockchain

# A Simple Token

```
3
4 // This is the main building block for smart contracts.
5 contract SimpleToken {
6     // Some string type variables to identify the token.
7     string public name = "My D21 Token";
8     string public symbol = "D21";
9
10    // The fixed amount of tokens, stored in an unsigned integer type variable.
11    uint256 public totalSupply = 1000000;
12
13    // A mapping is a key/value map. Here we store each account's balance.
14    mapping(address => uint256) balances;
15
16    // The Transfer event helps off-chain applications understand
17    // what happens within your contract.
18    event Transfer(address indexed _from, address indexed _to, uint256 _value);
19
20    /**
21     * Contract initialization.
22     */
23    constructor() {
24        // The totalSupply is assigned to the transaction sender, which is the
25        // account that is deploying the contract.
26        balances[msg.sender] = totalSupply;
27    }
```

# Simple Token Methods

```
29  /**
30  * A function to transfer tokens.
31  *
32  * The `external` modifier makes a function *only* callable from *outside*
33  * the contract.
34  */
35  function transfer(address to, uint256 amount) external {
36      // Check if the transaction sender has enough tokens.
37      // If `require`'s first argument evaluates to `false`, the
38      // transaction will revert.
39      require(balances[msg.sender] >= amount, "Not enough tokens");
40
41      // Transfer the amount.
42      balances[msg.sender] -= amount;
43      balances[to] += amount;
44
45      // Notify off-chain applications of the transfer.
46      emit Transfer(msg.sender, to, amount);
47  }
48
49  /**
50  * Read only function to retrieve the token balance of a given account.
51  *
52  * The `view` modifier indicates that it doesn't modify the contract's
53  * state, which allows us to call it without executing a transaction.
54  */
55  function balanceOf(address account) external view returns (uint256) {
56      return balances[account];
57  }
```

# The need for a Token Standard

- Problem : the token interface might change from one contract to another
- ✓ Having a standard token interface create predictable APIs for wallets, dApps, and protocols

# Ethereum Standards for Digital Assets



## Ethereum Token Standards

### ERC - 20



#### Fungible Tokens

Most basic token standard, used to create interchangeable tokens

Trade-able virtual currencies  
Governance/voting tokens  
Staking tokens



WBTC



USDC



CRV

### ERC - 721



#### Non-Fungible Tokens

Basic NFT standard, used to create unique tokens, distinguishable from others in the same collection

Collectable art  
Digital items and property  
Tickets (events, seats, lottery)



CryptoPunks

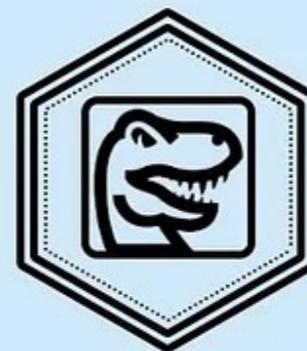


WMVG



Audioglyph

### ERC - 1155



#### Multi-Token Standard

A single interface that manages any combination of multiple token types (fungible, non-fungible, etc).

Alternate to ERC-20 and ERC-721  
Video game items  
Memorabilia



Enjin Tokens

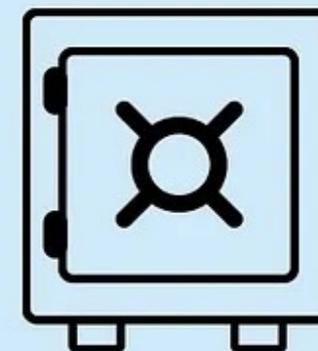


Beanstalk  
Fertilizer



Adidas  
Metaverse

### ERC - 4626



#### Tokenized Vault Standard

A standard that represents a yield-bearing vault; extending ERC-20 to include deposit, redeem, etc

Lending markets  
Interest bearing tokens  
Aggregators



Yearn  
v3 Vaults



Umami  
USDC Vault



EnreachDAO  
Yaggr (BNB)

Twitter: @SalomonCrypto

# ERC-20 Fungible Tokens

Tokens are fungible meaning they are interchangeable

➔ Represent the ownership of a quantity relative to a total supply

Features:

- Allow token holder owner to transfer to a address
- Or allow the holder to delegate spending to another address via allowances (e.g. useful to build an exchange)

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```



Interface of the ERC-20 standard as defined in the ERC.

## Functions

- [totalSupply\(\)](#)
- [balanceOf\(account\)](#)
- [transfer\(to, value\)](#)
- [allowance\(owner, spender\)](#)
- [approve\(spender, value\)](#)
- [transferFrom\(from, to, value\)](#)

## Events

- [Transfer\(from, to, value\)](#)
- [Approval\(owner, spender, value\)](#)

# ERC20

# ERC20 Applications

Stablecoins (USDC, DAI)

Governance tokens

Utility/Incentive tokens

Wrapped assets (WETH-style patterns)

# ERC721 Non-Fungible Token (NFT)

Tokens are non-fungible meaning they are all unique and identified by an id

➔ Represents the ownership of a unique asset within a collection

## Features

- Allow a token holder to transfer ownership of the asset
- Or allow the owner to delegate transfer to another address via an approval (e.g. useful to build a marketplace)

# ERC 721

```
import "@openzeppelin/contracts/token/ERC721/IERC721.sol";
```



Required interface of an ERC-721 compliant contract.

## Functions

- [balanceOf\(owner\)](#)
- [ownerOf\(tokenId\)](#)
- [safeTransferFrom\(from, to, tokenId, data\)](#)
- [safeTransferFrom\(from, to, tokenId\)](#)
- [transferFrom\(from, to, tokenId\)](#)
- [approve\(to, tokenId\)](#)
- [setApprovalForAll\(operator, approved\)](#)
- [getApproved\(tokenId\)](#)
- [isApprovedForAll\(owner, operator\)](#)

## Events

- [Transfer\(from, to, tokenId\)](#)
- [Approval\(owner, approved, tokenId\)](#)
- [ApprovalForAll\(owner, operator, approved\)](#)

# The OpenSea Metadata Standard

