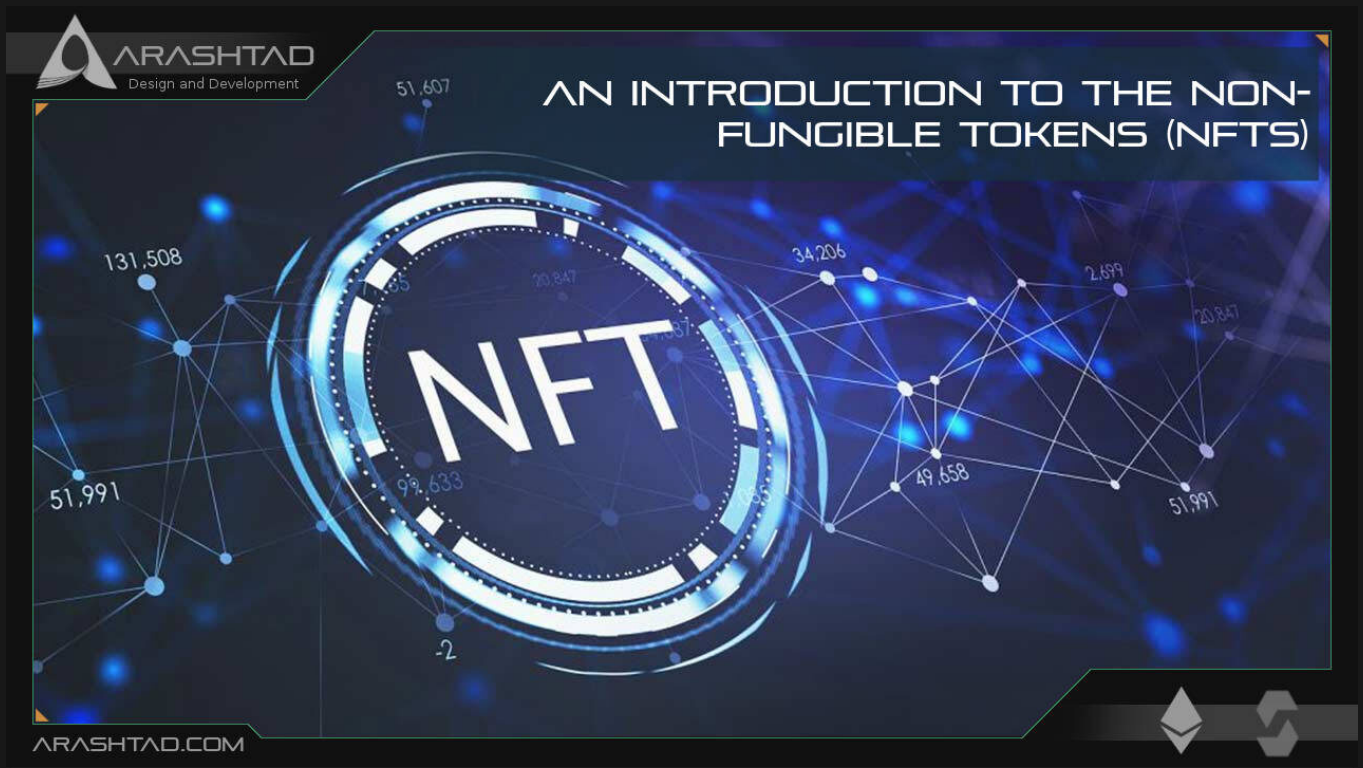
# An Introduction to the Non-Fungible Tokens (NFTs)

No comments




***In this tutorial, we are going to introduce the Non-Fungible Tokens or ERC-721 standard. In addition to that, we are going to get familiar with the interface of ERC-721 so that we can later use them in NFT deployments. Every NFT project also needs an ERC-721 JSON metadata that we are going to introduce in different sections.***

## Introduction to Non-Fungible Tokens

Non-Fungible or ERC-721 tokens are the kind of tokens that have unique features in other words, they cannot be exchanged by one another because they are all identified as unique. The use cases for NFTs are platforms that offer collectible items (like artworks), lottery tickets, numbered seats for cinemas, concerts, access keys, and so on.

**ERC-721 Interface**

As mentioned earlier, the NFTs follow the ERC-721 standard. Here we take a look at the ERC-721 interface, which we are going to use in the next parts of our NFT tutorials when we want to write a contract for our collectibles:

```
pragma solidity ^ 0.4.20;

interface ERC721 {
    event Transfer(address indexed _from, address indexed _to, uint256
 indexed _tokenId);
}
```

The above event called Transfer emits when ownership of any NFT changes by any mechanism. This event emits when NFTs are created (`from` == 0) and destroyed (`to` == 0).
Exception: during contract creation, any number of NFTs may be created and assigned without emitting Transfer. At the time of any transfer, the approved address for that NFT (if any) is reset to none.

```
event Approval(address indexed _owner, address indexed _approved,
uint256 indexed _tokenId);
```

The above Approval event emits when the approved address for an NFT is changed or reaffirmed. The zero address indicates there is no approved address. When a Transfer event emits, this also indicates that the approved address for that NFT (if any) is reset to none.

```
event ApprovalForAll(address indexed _owner, address indexed _operator
, bool _approved);
```

ApprovalForAll event emits when an operator is enabled or disabled for an owner. The operator can manage all NFTs of the owner.

```
function balanceOf(address _owner) external view returns (uint256);
```

The above function counts all NFTs assigned to an owner. NFTs assigned to the zero address is considered invalid, and this function throws for queries about the zero address. This function returns the number of NFTs owned by `_owner`, possibly zero.
_owner = An address for whom to query the balance

```
function ownerOf(uint256 _tokenId) external view returns (address);
```

The above function finds the owner of an NFT. The NFTs that are assigned to zero address are considered invalid, and

queries about them do throw. `_tokenId` is the identifier for an NFT. The function returns the address of the owner of the NFT.

```
function safeTransferFrom(address _from, address _to, uint256 _tokenId
, bytes data) external payable;
```

The above function transfers the ownership of an NFT from one address to another address. It throws an error in the following cases:

1. `msg.sender` is not the current owner, an authorized operator, or the approved address for this NFT.
2. `_from` is not the current owner.
3. `_to` is the zero address.
4. `_tokenId` is not a valid NFT.

When transfer is complete, this function checks if `_to` is a smart contract (code size > 0). If so, it calls `onERC721Received` on `_to` and throws an error if the return value is not:

```
`bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"
))`.
```

The attributes of the function are as follows:

1. `_from` The current owner of the NFT
2. `_to` The new owner
3. `_tokenId` The NFT to transfer
4. `_data` Additional data with no specified format, sent in the call to `_to`

```
function safeTransferFrom(address _from, address _to, uint256 _tokenId
) external payable;
```

The above function transfers the ownership of an NFT from one address to another. This function works the same as the previous one with an extra data parameter, except that this function just sets data to "".
The attributes of this function are as follows:

1. `_from` The current owner of the NFT
2. `_to` The new owner
3. `_tokenId` The NFT to transfer

```
function transferFrom(address _from, address _to, uint256 _tokenId)
external payable;
```

The above function transfers ownership of an NFT. The caller is responsible to confirm that `_to` is capable of receiving NFTs otherwise they may be permanently lost. It throws an error if:

1. `msg.sender` is not the current owner, an authorized operator, or the approved address for this NFT.
2. `_from` is not the current owner.
3. `_to` is the zero address.
4. `_tokenId` is not a valid NFT.

The attributes of this function are:

1. `_from` The current owner of the NFT
2. `_to` The new owner
3. `_tokenId` The NFT to transfer

```
function approve(address _approved, uint256 _tokenId) external payable
;
```

The above function changes or reaffirms the approved address for an NFT. The zero address indicates there is no approved address. It throws an error if:
`msg.sender` is not the current NFT owner or an authorizedoperator of the current owner.
The attributes are:

1. `_approved`: The new approved NFT controller
2. `_tokenId`: The NFT to approve

```
function setApprovalForAll(address _operator, bool _approved) external
;
```

The above function enables or disables the approval for a third party ("operator") to manage all of `msg.sender` 's assets. Emits the ApprovalForAll event. The contract MUST allow multiple operators per owner.
The attributes are:

1. `_operator`: Address to add to the set of authorized operators
2. `_approved`: True if the operator is approved, false to revoke approval

```
function getApproved(uint256 _tokenId) external
 view returns (address);
```

The above function gets the approved address for a single NFT. Throws error if `_tokenId` is not a valid NFT.
The function attributes are:

1. `_tokenId` The NFT to find the approved address for

The function returns:
The approved address for this NFT or the zero address if there is none otherwise.

```
function isApprovedForAll(address _owner, address _operator) external
 view returns (bool);
```

The above function queries if an address is an authorized operator for another address.The parameters of this function are:

1. `_owner`: The address that owns the NFTs
2. `_operator`: The address that acts on behalf of the owner

The function returns:
True if `_operator` is an approved operator for `_owner`, false.

```
interface ERC165 {
            function supportsInterface(bytes4 interfaceID) external
 view returns (bool);
}
```

The above function is related to the ERC-165 interface and queries if a contract implements an interface. Interface identification is specified in ERC-165. This function uses less than 30,000 gas.The attributes of this function are:

1. Param interfaceID The interface identifier, as specified in ERC-165.

The function returns:
`true` if the contract implements `interfaceID` and `interfaceID` is not `0xffffffff`, `false` otherwise.

**ERC-721 Metadata**

We also have an ERC721 JSON metadata of each NFT in the following format which specifies the features of the NFT. And we are going to use it in our future NFT projects.

```
{
    "title": "Asset Metadata",
    "type": "object",
    "properties": {
        "name": {
            "type": "string",
            "description":
"Identifies the asset to which this NFT represents"
        },
        "description": {
            "type": "string",
            "description":
"Describes the asset to which this NFT represents"
        },
        "image": {
            "type": "string",
            "description":
"A URI pointing to a resource with mime type image/* representing the asset t
                        which this NFT represents. Consider making any image
                        320 and 1080 pixels and aspect ratio between 1.91:1

        }
    }
}
```

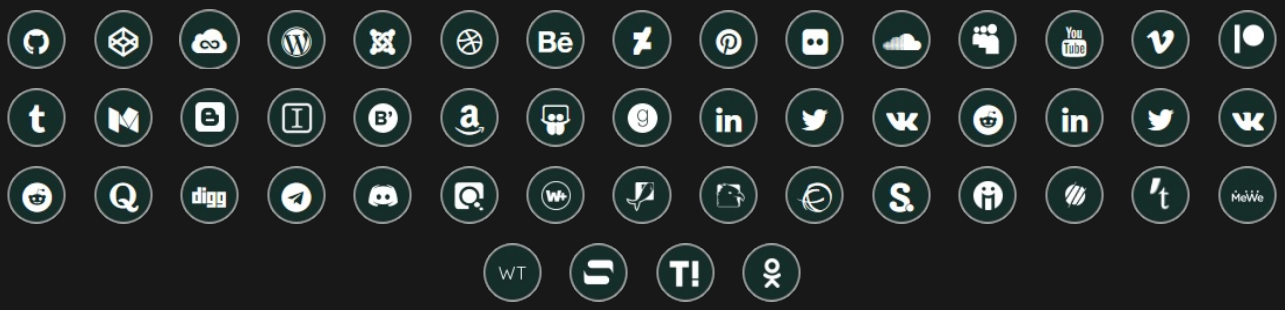
### Last Consideration

In this article, we have introduced the ERC-721 standard for NFTs (Non-fungible tokens). Furthermore, we have taken a deeper look at the ERC-721 interface functions and attributes in addition to its JSON metadata format.

# Join Arashtad Community

## Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.



## Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these benefitial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

SIGN UP | NEWSLETTER | RSS FEED