

Standards Track: ERC

ERC-721: Non-Fungible Token Standard

Authors William Entriken (@fulldecent), Dieter Shirley <dete@axiomzen.co>, Jacob Evans <jacob@dekz.net>, Nastassia Sachs <nastassia.sachs@protonmail.com>

Created 2018-01-24

Requires EIP-165

Table of Contents

- Simple Summary
- Abstract
- Motivation
- Specification
 - Caveats
- Rationale
- Backwards Compatibility
- Test Cases
- Implementations
- References
- Copyright

Simple Summary

A standard interface for non-fungible tokens, also known as deeds.

Abstract

The following standard allows for the implementation of a standard API for NFTs within smart contracts. This standard provides basic functionality to track and transfer NFTs.

We considered use cases of NFTs being owned and transacted by individuals as well as consignment to third party brokers/wallets/auctioneers (“operators”). NFTs can represent ownership over digital or physical assets. We considered a diverse universe of assets, and we know you will dream up many more:

- Physical property — houses, unique artwork
- Virtual collectibles — unique pictures of kittens, collectible cards
- “Negative value” assets — loans, burdens and other responsibilities

In general, all houses are distinct and no two kittens are alike. NFTs are *distinguishable* and you must track the ownership of each one separately.

Motivation

A standard interface allows wallet/broker/auction applications to work with any NFT on Ethereum. We provide for simple ERC-721 smart contracts as well as contracts that track an *arbitrarily large* number of NFTs. Additional applications are discussed below.

This standard is inspired by the ERC-20 token standard and builds on two years of experience since EIP-20 was created. EIP-20 is insufficient for tracking NFTs because each asset is distinct (non-fungible) whereas each of a quantity of tokens is identical (fungible).

Differences between this standard and EIP-20 are examined below.

Specification

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Every ERC-721 compliant contract must implement the `ERC721` and `ERC165` interfaces (subject to "caveats" below):

```
pragma solidity ^0.4.20;

/// @title ERC-721 Non-Fungible Token Standard
/// @dev See https://eips.ethereum.org/EIPS/eip-721
/// Note: the ERC-165 identifier for this interface is 0x80ac58cd.
interface ERC721 /* is ERC165 */ {
    /// @dev This 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 Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);

    /// @dev This 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 Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);

    /// @dev This emits when an operator is enabled or disabled for an owner.
    /// The operator can manage all NFTs of the owner.
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);

    /// @notice Count all NFTs assigned to an owner
    /// @dev NFTs assigned to the zero address are considered invalid, and this
    /// function throws for queries about the zero address.
    /// @param _owner An address for whom to query the balance
    /// @return The number of NFTs owned by `_owner`, possibly zero
    function balanceOf(address _owner) external view returns (uint256);

    /// @notice Find the owner of an NFT
```

```

/// @dev NFTs assigned to zero address are considered invalid, and queries
/// about them do throw.
/// @param _tokenId The identifier for an NFT
/// @return The address of the owner of the NFT
function ownerOf(uint256 _tokenId) external view returns (address);

/// @notice Transfers the ownership of an NFT from one address to another address
/// @dev Throws unless `msg.sender` is the current owner, an authorized
/// operator, or the approved address for this NFT. Throws if `_from` is
/// not the current owner. Throws if `_to` is the zero address. Throws if
/// `_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 if the return value is not
/// `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`.
/// @param _from The current owner of the NFT
/// @param _to The new owner
/// @param _tokenId The NFT to transfer
/// @param data Additional data with no specified format, sent in call to `_to`
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external

/// @notice Transfers the ownership of an NFT from one address to another address
/// @dev This works identically to the other function with an extra data parameter,
/// except this function just sets data to "".
/// @param _from The current owner of the NFT
/// @param _to The new owner
/// @param _tokenId The NFT to transfer
function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;

/// @notice Transfer ownership of an NFT -- THE CALLER IS RESPONSIBLE
/// TO CONFIRM THAT `_to` IS CAPABLE OF RECEIVING NFTS OR ELSE
/// THEY MAY BE PERMANENTLY LOST
/// @dev Throws unless `msg.sender` is the current owner, an authorized
/// operator, or the approved address for this NFT. Throws if `_from` is
/// not the current owner. Throws if `_to` is the zero address. Throws if
/// `_tokenId` is not a valid NFT.
/// @param _from The current owner of the NFT
/// @param _to The new owner
/// @param _tokenId The NFT to transfer
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;

/// @notice Change or reaffirm the approved address for an NFT
/// @dev The zero address indicates there is no approved address.
/// Throws unless `msg.sender` is the current NFT owner, or an authorized
/// operator of the current owner.
/// @param _approved The new approved NFT controller
/// @param _tokenId The NFT to approve
function approve(address _approved, uint256 _tokenId) external payable;

/// @notice Enable or disable approval for a third party ("operator") to manage
/// all of `msg.sender`'s assets
/// @dev Emits the ApprovalForAll event. The contract MUST allow

```

```

/// multiple operators per owner.
/// @param _operator Address to add to the set of authorized operators
/// @param _approved True if the operator is approved, false to revoke approval
function setApprovalForAll(address _operator, bool _approved) external;

/// @notice Get the approved address for a single NFT
/// @dev Throws if `_tokenId` is not a valid NFT.
/// @param _tokenId The NFT to find the approved address for
/// @return The approved address for this NFT, or the zero address if there is none
function getApproved(uint256 _tokenId) external view returns (address);

/// @notice Query if an address is an authorized operator for another address
/// @param _owner The address that owns the NFTs
/// @param _operator The address that acts on behalf of the owner
/// @return True if `_operator` is an approved operator for `_owner`, false otherwise
function isApprovedForAll(address _owner, address _operator) external view returns (bool);
}

interface ERC165 {
    /// @notice Query if a contract implements an interface
    /// @param interfaceID The interface identifier, as specified in ERC-165
    /// @dev Interface identification is specified in ERC-165. This function
    /// uses less than 30,000 gas.
    /// @return `true` if the contract implements `interfaceID` and
    /// `interfaceID` is not 0xffffffff, `false` otherwise
    function supportsInterface(bytes4 interfaceID) external view returns (bool);
}

```

A wallet/broker/auction application **MUST** implement the **wallet interface** if it will accept safe transfers.

```

/// @dev Note: the ERC-165 identifier for this interface is 0x150b7a02.
interface ERC721TokenReceiver {
    /// @notice Handle the receipt of an NFT
    /// @dev The ERC721 smart contract calls this function on the recipient
    /// after a `transfer`. This function MAY throw to revert and reject the
    /// transfer. Return of other than the magic value MUST result in the
    /// transaction being reverted.
    /// Note: the contract address is always the message sender.
    /// @param _operator The address which called `safeTransferFrom` function
    /// @param _from The address which previously owned the token
    /// @param _tokenId The NFT identifier which is being transferred
    /// @param _data Additional data with no specified format
    /// @return `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    /// unless throwing
    function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes _data) external returns (bytes4);
}

```

The **metadata extension** is **OPTIONAL** for ERC-721 smart contracts (see "caveats", below). This allows your smart contract to be interrogated for its name and for details about the assets which your NFTs represent.

```

/// @title ERC-721 Non-Fungible Token Standard, optional metadata extension
/// @dev See https://eips.ethereum.org/EIPS/eip-721
/// Note: the ERC-165 identifier for this interface is 0x5b5e139f.
interface ERC721Metadata /* is ERC721 */ {
    /// @notice A descriptive name for a collection of NFTs in this contract
    function name() external view returns (string _name);

    /// @notice An abbreviated name for NFTs in this contract
    function symbol() external view returns (string _symbol);

    /// @notice A distinct Uniform Resource Identifier (URI) for a given asset.
    /// @dev Throws if `_tokenId` is not a valid NFT. URIs are defined in RFC
    /// 3986. The URI may point to a JSON file that conforms to the "ERC721
    /// Metadata JSON Schema".
    function tokenURI(uint256 _tokenId) external view returns (string);
}

```

This is the "ERC721 Metadata JSON Schema" referenced above.

```

{
  "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"
    }
  }
}

```

The **enumeration extension** is OPTIONAL for ERC-721 smart contracts (see "caveats", below). This allows your contract to publish its full list of NFTs and make them discoverable.

```

/// @title ERC-721 Non-Fungible Token Standard, optional enumeration extension
/// @dev See https://eips.ethereum.org/EIPS/eip-721
/// Note: the ERC-165 identifier for this interface is 0x780e9d63.
interface ERC721Enumerable /* is ERC721 */ {
    /// @notice Count NFTs tracked by this contract
    /// @return A count of valid NFTs tracked by this contract, where each one of
    /// them has an assigned and queryable owner not equal to the zero address
    function totalSupply() external view returns (uint256);
}

```

```

/// @notice Enumerate valid NFTs
/// @dev Throws if `_index` >= `totalSupply()`.
/// @param _index A counter less than `totalSupply()`
/// @return The token identifier for the `_index`th NFT,
/// (sort order not specified)
function tokenByIndex(uint256 _index) external view returns (uint256);

/// @notice Enumerate NFTs assigned to an owner
/// @dev Throws if `_index` >= `balanceOf(_owner)` or if
/// `_owner` is the zero address, representing invalid NFTs.
/// @param _owner An address where we are interested in NFTs owned by them
/// @param _index A counter less than `balanceOf(_owner)`
/// @return The token identifier for the `_index`th NFT assigned to `_owner`,
/// (sort order not specified)
function tokenOfOwnerByIndex(address _owner, uint256 _index) external view returns (uint256);
}

```

Caveats

The 0.4.20 Solidity interface grammar is not expressive enough to document the ERC-721 standard. A contract which complies with ERC-721 MUST also abide by the following:

- Solidity issue #3412: The above interfaces include explicit mutability guarantees for each function. Mutability guarantees are, in order weak to strong: `payable`, implicit nonpayable, `view`, and `pure`. Your implementation MUST meet the mutability guarantee in this interface and you MAY meet a stronger guarantee. For example, a `payable` function in this interface may be implemented as nonpayable (no state mutability specified) in your contract. We expect a later Solidity release will allow your stricter contract to inherit from this interface, but a workaround for version 0.4.20 is that you can edit this interface to add stricter mutability before inheriting from your contract.
- Solidity issue #3419: A contract that implements `ERC721Metadata` or `ERC721Enumerable` SHALL also implement `ERC721`. ERC-721 implements the requirements of interface ERC-165.
- Solidity issue #2330: If a function is shown in this specification as `external` then a contract will be compliant if it uses `public` visibility. As a workaround for version 0.4.20, you can edit this interface to switch to `public` before inheriting from your contract.
- Solidity issues #3494, #3544: Use of `this.*.selector` is marked as a warning by Solidity, a future version of Solidity will not mark this as an error.

If a newer version of Solidity allows the caveats to be expressed in code, then this EIP MAY be updated and the caveats removed, such will be equivalent to the original specification.

Rationale

There are many proposed uses of Ethereum smart contracts that depend on tracking distinguishable assets. Examples of existing or planned NFTs are LAND in Decentraland, the eponymous punks in CryptoPunks, and in-game items using systems like DMarket or EnjinCoin. Future uses include tracking real-world assets, like real-estate (as envisioned by companies like Ubitquity or Propy). It is critical in each of these cases that

these items are not “lumped together” as numbers in a ledger, but instead each asset must have its ownership individually and atomically tracked. Regardless of the nature of these assets, the ecosystem will be stronger if we have a standardized interface that allows for cross-functional asset management and sales platforms.

“NFT” Word Choice

“NFT” was satisfactory to nearly everyone surveyed and is widely applicable to a broad universe of distinguishable digital assets. We recognize that “deed” is very descriptive for certain applications of this standard (notably, physical property).

Alternatives considered: distinguishable asset, title, token, asset, equity, ticket

NFT Identifiers

Every NFT is identified by a unique `uint256` ID inside the ERC-721 smart contract. This identifying number SHALL NOT change for the life of the contract. The pair `(contract address, uint256 tokenId)` will then be a globally unique and fully-qualified identifier for a specific asset on an Ethereum chain. While some ERC-721 smart contracts may find it convenient to start with ID 0 and simply increment by one for each new NFT, callers SHALL NOT assume that ID numbers have any specific pattern to them, and MUST treat the ID as a “black box”. Also note that NFTs MAY become invalid (be destroyed). Please see the enumeration functions for a supported enumeration interface.

The choice of `uint256` allows a wide variety of applications because UUIDs and sha3 hashes are directly convertible to `uint256`.

Transfer Mechanism

ERC-721 standardizes a safe transfer function `safeTransferFrom` (overloaded with and without a `bytes` parameter) and an unsafe function `transferFrom`. Transfers may be initiated by:

- The owner of an NFT
- The approved address of an NFT
- An authorized operator of the current owner of an NFT

Additionally, an authorized operator may set the approved address for an NFT. This provides a powerful set of tools for wallet, broker and auction applications to quickly use a *large* number of NFTs.

The transfer and accept functions’ documentation only specify conditions when the transaction MUST throw. Your implementation MAY also throw in other situations. This allows implementations to achieve interesting results:

- **Disallow transfers if the contract is paused** — prior art, CryptoKitties deployed contract, line 611
- **Blocklist certain address from receiving NFTs** — prior art, CryptoKitties deployed contract, lines 565, 566
- **Disallow unsafe transfers** — `transferFrom` throws unless `_to` equals `msg.sender` or `countOf(_to)` is non-zero or was non-zero previously (because such cases are safe)
- **Charge a fee to both parties of a transaction** — require payment when calling `approve` with a non-zero `_approved` if it was previously the zero address, refund payment if calling `approve` with

the zero address if it was previously a non-zero address, require payment when calling any transfer function, require transfer parameter `_to` to equal `msg.sender`, require transfer parameter `_to` to be the approved address for the NFT

- **Read only NFT registry** — always throw from `safeTransferFrom`, `transferFrom`, `approve` and `setApprovalForAll`

Failed transactions will throw, a best practice identified in ERC-223, ERC-677, ERC-827 and OpenZeppelin's implementation of SafeERC20.sol. ERC-20 defined an `allowance` feature, this caused a problem when called and then later modified to a different amount, as on OpenZeppelin issue #438. In ERC-721, there is no allowance because every NFT is unique, the quantity is none or one. Therefore we receive the benefits of ERC-20's original design without problems that have been later discovered.

Creation of NFTs ("minting") and destruction of NFTs ("burning") is not included in the specification. Your contract may implement these by other means. Please see the `event` documentation for your responsibilities when creating or destroying NFTs.

We questioned if the `operator` parameter on `onERC721Received` was necessary. In all cases we could imagine, if the operator was important then that operator could transfer the token to themselves and then send it – then they would be the `from` address. This seems contrived because we consider the operator to be a temporary owner of the token (and transferring to themselves is redundant). When the operator sends the token, it is the operator acting on their own accord, NOT the operator acting on behalf of the token holder. This is why the operator and the previous token owner are both significant to the token recipient.

Alternatives considered: only allow two-step ERC-20 style transaction, require that transfer functions never throw, require all functions to return a boolean indicating the success of the operation.

ERC-165 Interface

We chose Standard Interface Detection (ERC-165) to expose the interfaces that a ERC-721 smart contract supports.

A future EIP may create a global registry of interfaces for contracts. We strongly support such an EIP and it would allow your ERC-721 implementation to implement `ERC721Enumerable`, `ERC721Metadata`, or other interfaces by delegating to a separate contract.

Gas and Complexity (regarding the enumeration extension)

This specification contemplates implementations that manage a few and *arbitrarily large* numbers of NFTs. If your application is able to grow then avoid using for/while loops in your code (see CryptoKitties bounty issue #4). These indicate your contract may be unable to scale and gas costs will rise over time without bound.

We have deployed a contract, XXXXERC721, to Testnet which instantiates and tracks 340282366920938463463374607431768211456 different deeds (2^{128}). That's enough to assign every IPV6 address to an Ethereum account owner, or to track ownership of nanobots a few micron in size and in aggregate totalling half the size of Earth. You can query it from the blockchain. And every function takes less gas than querying the ENS.

This illustration makes clear: the ERC-721 standard scales.

Alternatives considered: remove the asset enumeration function if it requires a for-loop, return a Solidity array type from enumeration functions.

Privacy

Wallets/brokers/auctioneers identified in the motivation section have a strong need to identify which NFTs an owner owns.

It may be interesting to consider a use case where NFTs are not enumerable, such as a private registry of property ownership, or a partially-private registry. However, privacy cannot be attained because an attacker can simply (!) call `ownerOf` for every possible `tokenId`.

Metadata Choices (metadata extension)

We have required `name` and `symbol` functions in the metadata extension. Every token EIP and draft we reviewed (ERC-20, ERC-223, ERC-677, ERC-777, ERC-827) included these functions.

We remind implementation authors that the empty string is a valid response to `name` and `symbol` if you protest to the usage of this mechanism. We also remind everyone that any smart contract can use the same name and symbol as *your* contract. How a client may determine which ERC-721 smart contracts are well-known (canonical) is outside the scope of this standard.

A mechanism is provided to associate NFTs with URIs. We expect that many implementations will take advantage of this to provide metadata for each NFT. The image size recommendation is taken from Instagram, they probably know much about image usability. The URI MAY be mutable (i.e. it changes from time to time). We considered an NFT representing ownership of a house, in this case metadata about the house (image, occupants, etc.) can naturally change.

Metadata is returned as a string value. Currently this is only usable as calling from `web3`, not from other contracts. This is acceptable because we have not considered a use case where an on-blockchain application would query such information.

Alternatives considered: put all metadata for each asset on the blockchain (too expensive), use URL templates to query metadata parts (URL templates do not work with all URL schemes, especially P2P URLs), multiaddr network address (not mature enough)

Community Consensus

A significant amount of discussion occurred on the original ERC-721 issue, additionally we held a first live meeting on Gitter that had good representation and well advertised (on Reddit, in the Gitter #ERC channel, and the original ERC-721 issue). Thank you to the participants:

- @ImAllInNow Rob from DEC Gaming / Presenting Michigan Ethereum Meetup Feb 7
- @Arachnid Nick Johnson
- @jadhavajay Ajay Jadhav from AyanWorks
- @superphly Cody Marx Bailey - XRAM Capital / Sharing at hackathon Jan 20 / UN Future of Finance Hackathon.

- @fulldcent William Entriken

A second event was held at ETHDenver 2018 to discuss distinguishable asset standards (notes to be published).

We have been very inclusive in this process and invite anyone with questions or contributions into our discussion. However, this standard is written only to support the identified use cases which are listed herein.

Backwards Compatibility

We have adopted `balanceOf`, `totalSupply`, `name` and `symbol` semantics from the ERC-20 specification. An implementation may also include a function `decimals` that returns `uint8(0)` if its goal is to be more compatible with ERC-20 while supporting this standard. However, we find it contrived to require all ERC-721 implementations to support the `decimals` function.

Example NFT implementations as of February 2018:

- CryptoKitties – Compatible with an earlier version of this standard.
- CryptoPunks – Partially ERC-20 compatible, but not easily generalizable because it includes auction functionality directly in the contract and uses function names that explicitly refer to the assets as “punks”.
- Auctionhouse Asset Interface – The author needed a generic interface for the Auctionhouse DApp (currently ice-boxed). His “Asset” contract is very simple, but is missing ERC-20 compatibility, `approve()` functionality, and metadata. This effort is referenced in the discussion for EIP-173.

Note: “Limited edition, collectible tokens” like Curio Cards and Rare Pepe are *not* distinguishable assets. They’re actually a collection of individual fungible tokens, each of which is tracked by its own smart contract with its own total supply (which may be `1` in extreme cases).

The `onERC721Received` function specifically works around old deployed contracts which may inadvertently return `1 (true)` in certain circumstances even if they don’t implement a function (see Solidity DelegateCallReturnValue bug). By returning and checking for a magic value, we are able to distinguish actual affirmative responses versus these vacuous `true`s.

Test Cases

Oxcert ERC-721 Token includes test cases written using Truffle.

Implementations

Oxcert ERC721 – a reference implementation

- MIT licensed, so you can freely use it for your projects
- Includes test cases
- Active bug bounty, you will be paid if you find errors

Su Squares – an advertising platform where you can rent space and place images

- Complete the Su Squares Bug Bounty Program to seek problems with this standard or its implementation
- Implements the complete standard and all optional interfaces

ERC721ExampleDeed – an example implementation

- Implements using the OpenZeppelin project format

XXXXERC721, by William Entriken – a scalable example implementation

- Deployed on testnet with 1 billion assets and supporting all lookups with the metadata extension. This demonstrates that scaling is NOT a problem.

References

Standards

1. ERC-20 Token Standard.
2. ERC-165 Standard Interface Detection.
3. ERC-173 Owned Standard.
4. ERC-223 Token Standard.
5. ERC-677 `transferAndCall` Token Standard.
6. ERC-827 Token Standard.
7. Ethereum Name Service (ENS). <https://ens.domains>
8. Instagram – What's the Image Resolution? <https://help.instagram.com/1631821640426723>
9. JSON Schema. <https://json-schema.org/>
10. Multiaddr. <https://github.com/multiformats/multiaddr>
11. RFC 2119 Key words for use in RFCs to Indicate Requirement Levels. <https://www.ietf.org/rfc/rfc2119.txt>

Issues

1. The Original ERC-721 Issue. <https://github.com/ethereum/eips/issues/721>
2. Solidity Issue #2330 – Interface Functions are External. <https://github.com/ethereum/solidity/issues/2330>
3. Solidity Issue #3412 – Implement Interface: Allow Stricter Mutability. <https://github.com/ethereum/solidity/issues/3412>
4. Solidity Issue #3419 – Interfaces Can't Inherit. <https://github.com/ethereum/solidity/issues/3419>
5. Solidity Issue #3494 – Compiler Incorrectly Reasons About the `selector` Function. <https://github.com/ethereum/solidity/issues/3494>
6. Solidity Issue #3544 – Cannot Calculate Selector of Function Named `transfer`. <https://github.com/ethereum/solidity/issues/3544>
7. CryptoKitties Bounty Issue #4 – Listing all Kitties Owned by a User is $O(n^2)$. <https://github.com/axiomzen/cryptokitties-bounty/issues/4>
8. OpenZeppelin Issue #438 – Implementation of `approve` method violates ERC20 standard. <https://github.com/OpenZeppelin/zeppelin-solidity/issues/438>
9. Solidity DelegateCallReturnValue Bug. <https://solidity.readthedocs.io/en/develop/bugs.html#DelegateCallReturnValue>

Discussions

1. Reddit (announcement of first live discussion).
https://www.reddit.com/r/ethereum/comments/7r2ena/friday_119_live_discussion_on_erc_nonfungible/
2. Gitter #EIPs (announcement of first live discussion). <https://gitter.im/ethereum/EIPs?at=5a5f823fb48e8c3566f0a5e7>
3. ERC-721 (announcement of first live discussion).
<https://github.com/ethereum/eips/issues/721#issuecomment-358369377>
4. ETHDenver 2018. <https://ethdenver.com>

NFT Implementations and Other Projects

1. CryptoKitties. <https://www.cryptokitties.co>
2. 0xcert ERC-721 Token. <https://github.com/0xcert/ethereum-erc721>
3. Su Squares. <https://tenthousandsu.com>
4. Decentraland. <https://decentraland.org>
5. CryptoPunks. <https://www.larvalabs.com/cryptopunks>
6. DMarket. <https://www.dmarket.io>
7. Enjin Coin. <https://enjincoin.io>
8. Ubitquity. <https://www.ubitquity.io>
9. Propy. <https://tokensale.propy.com>
10. CryptoKitties Deployed Contract.
<https://etherscan.io/address/0x06012c8cf97bead5deae237070f9587f8e7a266d#code>
11. Su Squares Bug Bounty Program. <https://github.com/fulldecent/su-squares-bounty>
12. XXXXERC721. <https://github.com/fulldecent/erc721-example>
13. ERC721ExampleDeed. <https://github.com/nastassiasachs/ERC721ExampleDeed>
14. Curio Cards. <https://mycuriocards.com>
15. Rare Pepe. <https://rarepepewallet.com>
16. Auctionhouse Asset Interface.
<https://github.com/dob/auctionhouse/blob/master/contracts/Asset.sol>
17. OpenZeppelin SafeERC20.sol Implementation. <https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/SafeERC20.sol>

Copyright

Copyright and related rights waived via CC0.

Citation

Please cite this document as:

William Entriken (@fulldecent), Dieter Shirley <dete@axiomzen.co>, Jacob Evans <jacob@dekz.net>, Nastassia Sachs <nastassia.sachs@protonmail.com>, "ERC-721: Non-Fungible Token Standard," *Ethereum Improvement Proposals*, no. 721, January 2018. [Online serial]. Available: <https://eips.ethereum.org/EIPS/eip-721>.