# dGoods: Digital Goods Token Spec v0.1

Cameron Thacker* Stephan Cunningham* Rudy Koch*
John Linden*

## 1 Introduction

dGoods is an open source and free standard for handling the virtual representation of items, both digital and physical, on the EOS blockchain. While our initial focus is on assets for video games, we are designing something that does not preclude use for other types of goods. We believe this specification could become a standard when there is a need for supporting non-fungible, semi-fungible, and or fungible tokens on the EOS blockchain.

Our ansatz is that a game or company will want to create many different tokens that live under one main token symbol. Some of these tokens may be fungible or semi-fungible while others are non-fungible. What really separates this standard from other NFT standards that exist is the hierarchical naming structure of the token. Fungible tokens are identified by symbol:category:name and NFTs have an extra token id associated. This allows for unprecedented organization of tokens. It also enables wallets and dApps to surface tokens by category or name, providing search and filtering functionality.

Our design philosophy is to start small, remain flexible, and provide a robust digital goods standard that supports a diverse and innovative development community. Therefore, much of this specification is a logical extension of the standard EOS token with the addition of significant functionality improvements that will allow teams to easily integrate and display virtual items. In the following sections we describe the minimum set of required methods and functionality for implementing this specification, along with the table structure, and finally some example metadata templates.

## 2 Required Methods

**CREATE**: The create method instantiates a token. This is required before any tokens can be issued and sets properties such as the category, name, maximum supply, who has the ability to issue tokens, and if the token is fungible or not. Additionally, the first time this is called the symbol is recorded and subsequent calls must specify that symbol. Symbol must be A-Z, 7 character max. Name type is a string 12 characters max a-z, 1-5.

*Mythical Games

```
ACTION create(name issuer, name category, name token_name, bool fungible, bool
burnable, bool transferable, int64_t max_supply);
```

**ISSUE**: The issue method mints a token and gives ownership to the 'to' account name. For a valid call the symbol, category, and token_name must have been first created. Quantity must be equal to 1 if non-fungible or semi-fungible, otherwise quantity must be greater or equal to 0.0001.

```
ACTION issue(name to, name category, name token_name, double quantity, string
    metadata_uri, string memo);
```

**PAUSEXFER**: Pauses all transfers of all tokens. Only callable by the contract. If pause is true, will pause. If pause is false will unpause transfers.

```
ACTION pausexfer(bool pause);
```

**BURNNFT**: Burn method destroys specified tokens and frees the RAM. Only owner may call burn function and burnable must be true.

```
ACTION burnnft(name owner, vector<uint64_t> tokeninfo_ids);
```

**BURN**: Burn method destroys fungible tokens and frees the RAM if all are deleted. Only owner may call Burn function and burnable must be true.

```
ACTION burn(name owner, uint64_t global_id, double quantity);
```

**TRANSFERNFT**: Used to transfer non-fungible tokens. This allows for the ability to batch send tokens in one function call by passing in a list of token ids. Only the token owner can successfully call this function and transferable must be true.

```
ACTION transfernft(name from, name to, vector<uint64_t> tokeninfo_ids, string
    memo);
```

**TRANSFER**: The standard transfer method is callable only on fungible tokens. Instead of specifying tokens individually, a token is specified by it's global_id followed by an amount desired to be sent.

```
ACTION transfer(name from, name to, uint64_t global_id, double quantity, string
    memo);
```

# 3 Token Data

## 3.1 Symbol Info Table

Singleton which holds the symbol for the contract and is set the first call of `create`. `global_id` is incremented each time `create` is successfully called

```
// scope is self
TABLE symbolinfo {
    string symbol;
    uint64_t global_id;
};
```

## 3.2 Token Stats Table

Ensures there can be only one category, token_name pair. Stores whether a given token is fungible or burnable and what the current and max supplies are. Info is written when a token is created.

```
// scope is category, then token_name is unique
TABLE tokenstats {
    bool   fungible;
    bool   burnable;
    bool   transferable;
    name   issuer;
    name   token_name;
    uint64_t global_id;
    uint64_t max_supply;
    double current_supply;

    uint64_t primary_key() const { return token_name.value; }
};
```

## 3.3 Token Info Table

This is the global list of non or semi-fungible tokens. Secondary indices provide search by owner, category, or token_name.

```
// scope is self
TABLE tokeninfo {
    uint64_t id;
    uint64_t serial_number;
    name owner;
    name category;
```

```
    name token_name;
    string metadata_uri;

    uint64_t primary_key() const { return id; }
    uint64_t get_owner() const { return owner.value; }
};
```

### 3.4 Category Table

Holds all category names for easy querying.

```
// scope is self
TABLE categoryinfo {
  name category;

  uint64_t primary_key() const { return category.value; }
};
```

### 3.5 Account Table

The Account table holds the fungible tokens for an account, and a reference to how many NFTs that account owns of a given type.

```
// scope is owner
TABLE account {
    name category;
    name token_name;
    uint64_t global_id;
    double amount;

    uint64_t primary_key() const { return global_id; }
};
```

## 4 Metadata Templates

In order for wallets or dApps to support various digital goods, there need to be standards associated with the metadata. Our approach is to define templates based on the type of good. The following templates are candidates we have put forth, but this is to be a collaborative exercise. We want to provide a repository of templates that are agreed upon by the community. All metadata is formatted in JSON and must specify the `type` of template.

### 4.1 Type: 3dgameAsset

```
{
// Required Fields
"name": string; identifies the asset the token represents
"description": string; short description of the asset the token represents
"imageSmall": URI pointing to image resource size 150 x 150
"imageLarge": URI pointing to image resource size 1024 x 1024
"3drender": URI pointing to js webgl for rendering 3d object
"details": Key Value pairs to render in a detail view, could be things like
    {"strength": 5}
// Optional Fields
"authenticityImage": URI pointing to resource with mime type image representing
    certificate of authenticity
}
```

### 4.2 Type: 2dgameAsset

```
{
// Required Fields
"name": string; identifies the asset the token represents
"description": string; short description of the asset the token represents
"imageSmall": URI pointing to image resource size 150 x 150
"imageLarge": URI pointing to image resource size 1024 x 1024
"details": Key Value pairs to render in a detail view, could be things like
    {"strength": 5}
// Optional Fields
"authenticityImage": URI pointing to resource with mime type image representing
    certificate of authenticity
}
```

### 4.3 Type: ticket

```
{
// Required Fields
"name": string; identifies the event the ticket is for
"description": string; short description of the event the ticket is for
"location": string; name of the location where the event is being held
"address": string; address of the location where the event is being held
"date": Unix time; starting date of the event
"time": Unix time; starting time of the event
"imageSmall": URI pointing to image resource size 150 x 150
```

```
"imageLarge": URI pointing to image resource size 1024 x 1024
"details": Key Value pairs to render in a detail view, could be things like
    {"openingAct": "Nickelback"}
// Optional Fields
"authenticityImage": URI pointing to resource with mime type image representing
    certificate of authenticity
"duration": string; length of time the event lasts for
"row": string; row where seat is located
"seat": string; seat number or GA for General Admission
}
```

## 4.4 Type: art

```
{
// Required Fields
"name": string; identifies the asset the token represents
"description": string; short description of the asset the token represents
"creator": string; creator(s) of the art
"imageSmall": URI pointing to image resource size 150 x 150
"imageLarge": URI pointing to image resource size 1024 x 1024
"date": unix time; date artwork was created
"details": Key Value pairs to render in a detail view, could be things like
    {"materials": ["oil", "wood"], "location": "Norton Simon Museum"}
// Optional Fields
"authenticityImage": URI pointing to resource with mime type image representing
    certificate of authenticity or signature
}
```

## 4.5 Type: jewelry

```
{
//Required Fields
"name": string; identifies the watch the token represents
"description": string; short description of the watch the token represents
"imageSmall": URI pointing to image resource size 150 x 150
"imageLarge": URI pointing to image resource size 1024 x 1024
"image360": URI pointing to image resource for 360 image
"manufactureDate": Unix time; date watch was manufactured
"manufacturePlace": string; place watch was manufactured
"details": Key Value pairs to render in a detail view, could be things like
    {"caret": 1}
//Optional Fields
```

```
"authenticityImage": URI pointing to resource with mime type image representing
    certificate of authenticity
}
```

# 5 Definitions

- **RAM** - Refers to the permanent storage space on the EOS blockchain. Must be purchased from the EOS blockchain itself by sending EOS. Can also be sold if unused.
- **Account** - An EOS account is up to 12 characters and contains letters a-z, numbers 1-5 and a period (if using name service)
- **Symbol** - Defined to be a max of 7 characters capital letters A-Z only
- **Fungible** - in this context it refers to tokens which are interchangeable like EOS itself. When tokens are fungible, transferring any token is equivalent to any other.
- **Semi-fungible** - Tokens which are technically unique in that they have a unique identifier, but may share the same or similar data to other tokens. Many physical goods would fall into this category as they have a unique serial number but otherwise are indistinguishable.
- **Non-fungible** - Tokens that are unique