Steamodd Documentation

Release 4.22

Anthony Garcia

May 21, 2018

Contents

1 History **3**
- 1.1 Origin . . . 3
- 1.2 The name . . . 3

2 Installation **5**

3 Quick start **7**
- 3.1 Steam API key . . . 7
- 3.2 Components . . . 7

4 Steam API wrappers **9**
- 4.1 Low level methods . . . 9
- 4.2 High level methods . . . 9

5 Steam Inventory Manager **21**

6 VDF serializer **25**

Contents:

CHAPTER 1

History

Origin

Steamodd originated with an early version of OPTF2 which itself grew out of a 200 line script I wrote in the early days of the Steam API to find things I could complain about. Since then it has grown into a more and more capable and fully featured module with every version.

It is still a work in progress and the API is subject to change in breaking ways, however as of the 3.0 release I have began using a simple and meaningful versioning system that should make moving to new versions much easier. Major version numbers are incremented when the release makes breaking changes, minor version numbers are incremented when they are not. Meaning that it is safe to upgrade without having to change existing code.

The name

If there's one thing I've learned over the years and most recently from OPTF2 it's a good idea to record the meaning behind your project names if they aren't explicitly indicative of function or you *will* forget.

Steamodd quite simply stands for "Steam odds and ends". Even though it's starting to become more of a robust module it started out as a small and probably not very well designed script meant to be run as a tool instead of a reusable lib.

That's not to say that the name doesn't fit, since in addition to the strong implementation of the API it has the recent VDF support and the SIM layer to boast as useful but not exactly unrelated utilities.

CHAPTER 2

Installation

From command line:

```
$ pip install steamodd
```

If you wish to install it manually, Steamodd uses the standard distutils module. To install it run:

```
$ python setup.py install
```

For further instructions and commands run:

```
$ python setup.py --help
```

CHAPTER 3

Quick start

Steam API key

If you are going to use Steam API, you'll need to set Steam API key either from code:

```
>>> import steam
>>> steam.api.key.set(API_KEY)
```

Or set environmental variable:

```
$ export STEAMODD_API_KEY="your_key"
```

Most methods will not complete successfully without it. If you don't have an API key you can register for one on Steam.

Components

This library consists of three major components, which are documented separately:

- Steam API wrappers
- Steam Inventory Manager
- VDF serializer

CHAPTER 4

Steam API wrappers

Low level methods

You can call any method from any of Steam API interfaces using `steam.api.interface` class. Let's start with a quick example where we fetch user's game library.

Start by importing `interface` class:

```
>>> from steam.api import interface
```

Call method `GetOwnedGames` of interface `IPlayerService`. We are going to fetch games of user with id `76561198017493014` and include all application information:

```
>>> games = interface('IPlayerService').GetOwnedGames(steamid=76561198017493014, include_appinfo=1)
```

Since all method calls are lazy by default, this doesn't do anything at all. We'll need to either iterate over `games`, `print` it or access any of its dictionary keys:

```
>>> print(games['response']['game_count'])  # Fetches resource
249
```

Don't worry, resource isn't fetched each time you access results.

```
>>> print(games)  # Uses cached resource
{'response': {'games': [{'name': 'Counter-Strike', 'playtime_forever': 1570,...
```

You can disable laziness of `interface` by passing `aggressive=True` to its method:

```
>>> games = interface('IPlayerService').GetOwnedGames(steamid=76561198017493014, include_appinfo=1, a
```

You can also pass `since` (which translates to HTTP header `If-Modified-Since`) and `timeout` to method. By default, `version` is set to `1`. `data` can be passed to send POST data with requests. By default no data is assumed and request types are GET. Any number of additional keyword arguments are supported depending on the given method (see documentation).

High level methods

Following classes are convenience wrappers around *Low level methods*. `kwargs` are always passed to appropriate interface methods, so you can use all arguments from previous section.

Apps

class steam.apps.**app_list**(***kwargs*)

Retrieves a list of all Steam apps with their ID and localized name.

```
>>> from steam.apps import app_list
>>> app_list = app_list()
>>> 'Dota 2' in app_list
True
>>> 'Half-Life 3' in app_list
False
>>> len(app_list)
16762
>>> app_list['Counter-Strike']
(10, u'Counter-Strike')
```

Items

class steam.items.**schema**(*app*, *lang=None*, *version=1*, ***kwargs*)

Wrapper for item schema of certain games from Valve. Those are currently available (along with their ids):

- 260 - Counter Strike: Source Beta
- 440 - Team Fortress 2
- 520 - Team Fortress 2 Public Beta
- 570 - Dota 2
- 620 - Portal 2
- 710 - Counter-Strike: Global Offensive Beta Dev
- 816 - Dota 2 internal test
- 841 - Portal 2 Beta
- 205790 - Dota 2 (beta) test

Fetching schema of Team Fortress 2 (id 440) would look like:

```
>>> schema = steam.items.schema(440)
>>> schema[340].name
u'Defiant Spartan'
```

Schema class is an iterator of *steam.items.item()* objects. There are also other properties available:

client_url
Client schema URL

language
The ISO code of the language the instance is localized to

attributes
Returns all attributes in the schema

origins
Returns a map of all origins

qualities
Returns a dict of all possible qualities. The key(s) will be the ID, values are a tuple containing ID, name, localized name. To resolve a quality to a name intelligently use '_quality_definition'

particle_systems
Returns a dictionary of particle system dicts keyed by ID

kill_ranks
Returns a list of ranks for weapons with kill tracking

kill_types
Returns a dict with keys that are the value of the kill eater type attribute and values that are the name string

class steam.items.**item**(*item*, *schema=None*)
Stores a single inventory item.

This is a simple wrapper around JSON representation of both schema and inventory items. It is composed mostly from item properties:

```
>>> item = schema[340]
>>> item.name
u'Defiant Spartan'
>>> item.type
u'Hat'
>>> item.attributes
[<steam.items.item_attribute object at 0x10c8b3290>, <steam.items.item_attribute object at 0x10c
```

As convenience, item acts also as iterator of its attributes:

```
>>> for attribute in item.attributes:
...     attribute.name
...
u'kill eater score type'
u'kill eater kill type'
```

Following properties are available:

attributes
Returns a list of attributes

quality
Returns a tuple containing ID, name, and localized name of the quality

inventory_token
Returns the item's inventory token (a bitfield), deprecated.

position
Returns a position in the inventory or -1 if there's no position available (i.e. an item hasn't dropped yet or got displaced)

equipped
Returns a dict of classes that have the item equipped and in what slot

equipable_classes
Returns a list of classes that _can_ use the item.

schema_id
Returns the item's ID in the schema.

name
Returns the item's undecorated name

type
Returns the item's type. e.g. "Kukri" for the Tribalman's Shiv. If Valve failed to provide a translation the type will be the token without the hash prefix.

icon
URL to a small thumbnail sized image of the item, suitable for display in groups

image
URL to a full sized image of the item, for displaying 'zoomed-in' previews

id
Returns the item's unique serial number if it has one

original_id
Returns the item's original ID if it has one. This is the last "version" of the item before it was customized or otherwise changed

level
Returns the item's level (e.g. 10 for The Axtinguisher) if it has one

slot_name
Returns the item's slot as a string, this includes "primary", "secondary", "melee", and "head". Note that this is the slot of the item as it appears in the schema, and not necessarily the actual equipable slot. (see 'equipped')

cvar_class
Returns the item's class (what you use in the game to equip it, not the craft class)

craft_class
Returns the item's class in the crafting system if it has one. This includes hat, craft_bar, or craft_token.

craft_material_type

custom_name
Returns the item's custom name if it has one.

custom_description
Returns the item's custom description if it has one.

quantity
Returns the number of uses the item has, for example, a dueling mini-game has 5 uses by default

description
Returns the item's default description if it has one

min_level
Returns the item's minimum level (non-random levels will have the same min and max level)

contents
Returns the item in the container, if there is one. This will be a standard item object.

tradable
Somewhat of a WORKAROUND since this flag is there sometimes, "cannot trade" is there sometimes and then there's "always tradable". Opposed to only occasionally tradable when it feels like it. Attr 153 = cannot trade

craftable
Returns not craftable if the cannot craft flag exists. True, otherwise.

full_name
The full name of the item, generated depending on things such as its quality, rank, the schema language, and so on.

kill_eaters
Returns a list of tuples containing the proper localized kill eater type strings and their values according to set/type/value "order"

rank
Returns the item's rank (if it has one) as a dict that includes required score, name, and level.

available_styles
Returns a list of all styles defined for the item

style
The current style the item is set to or None if the item has no styles

capabilities
Returns a list of capabilities, these are flags for what the item can do or be done with

tool_metadata
A dict containing item dependant metadata such as holiday restrictions, types, and properties used by the client. Do not assume a stable syntax.

origin
Returns the item's localized origin name

class steam.items.**item_attribute**(*attribute*)
Wrapper around item attributes.

```
>>> for attribute in item.attributes:
...     print('%s: %s' % (attribute.name, attribute.formatted_value))
...
kill eater score type: 64.0
kill eater kill type: 64.0
```

Following properties are available:

formatted_value
Returns a formatted value as a string

formatted_description
Returns a formatted description string (%s* tokens replaced) or None if unavailable

name
The attribute's name

cvar_class
The attribute class, mostly non-useful except for console usage in some cases

id
The attribute ID, used for indexing the description blocks in the schema

type
Returns the attribute effect type (positive, negative, or neutral). This is not the same as the value type, see 'value_type'

value
Tries to intelligently return the raw value based on schema data. See also: 'value_int' and 'value_float'

value_int

value_float

description
Returns the attribute's description string, if it is intended to be printed with the value there will be a "%s1" token somewhere in the string. Use 'formatted_description' to build one automatically.

value_type
The attribute's type, note that this is the type of the attribute's value and not its affect on the item (i.e. negative or positive). See 'type' for that.

hidden
True if the attribute is "hidden" (not intended to be shown to the end user). Note that hidden attributes also usually have no description string

account_info
Certain attributes have a user's account information associated with it such as a gifted or crafted item.

A dict with two keys: 'persona' and 'id64'. None if the attribute has no account information attached to it.

class steam.items.**inventory**(*app*, *profile*, *schema=None*, ***kwargs*)
Wrapper around player inventory.

Fetches inventory of `player` for given `app` id:

```
>>> inventory = steam.items.inventory(570, 76561198017493014)
>>> for item in inventory:
...     item.name
...
'226749283'
'226749284'
```

Since inventory endpoint returns just very basic structure, we have to provide also `schema` if we want to work with fully populated *steam.items.item()* objects:

```
>>> schema = steam.items.schema(440)
>>> inventory = steam.items.inventory(440, 76561198017493014, schema)
>>> for item in inventory:
...     item.name
...
u'Mercenary'
u'Noise Maker - Winter Holiday'
```

There is also single property:

cells_total
The total number of cells in the inventory. This can be used to determine if the user has bought an expander. This is NOT the number of items in the inventory, but how many items CAN be stored in it. The actual current inventory size can be obtained by calling len on an inventory object

class steam.items.**assets**(*app*, *lang=None*, ***kwargs*)
Class for building asset catalogs

Fetches store assets for `app` id. Assets class acts as an iterator of *steam.items.asset_item()* objects.

```
>>> assets = steam.items.assets(440)
>>> for asset in assets:
...     asset.price
...
{u'MXN': 74.0, u'EUR': 4.59, u'VND': 109000.0, u'AUD': 6.5, ...}
{u'MXN': 112.0, u'EUR': 6.99, u'VND': 159000.0, u'AUD': 9.8, ...}
```

If you care only about single currency, `currency` keyword argument in ISO 4217 format is also accepted.

```
>>> assets = steam.items.assets(440, currency="RUB")
>>> for asset in assets:
...     asset.price
...
{u'RUB': 290.0}
{u'RUB': 435.0}
```

All available tags of assets are available in following property:

tags
Returns a dict that is a map of the internal tag names for this catalog to the localized labels.

class steam.items.**asset_item**(*asset*, *catalog*)
Stores a single item from a steam asset catalog

tags
Returns a dict containing tags and their localized labels as values

base_price
The price the item normally goes for, not including discounts.

price
Returns the most current price available, which may include sales/discounts

name
The asset "name" which is in fact a schema id of item.

Localization

class steam.loc.**language**(*code=None*)
Steam API localization tools and reference

```
>>> language = steam.loc.language('nl_NL')
>>> language.name
'Dutch'
>>> language.code
'nl_NL'
```

If language is not specified, it defaults to English:

```
>>> language = steam.loc.language()
>>> language.name
'English'
>>> language.code
'en_US'
```

If language isn't supported, __init__ raises *steam.loc.LanguageUnsupportedError()*

```
>>> language = steam.loc.language('sk_SK')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "steam/loc.py", line 68, in __init__
    raise LanguageUnsupportedError(code)
steam.loc.LanguageUnsupportedError: sk_sk
```

Properties:

code

name

class steam.loc.**LanguageUnsupportedError**

Remote storage

Tools for probing Steam's UGC file storage system. UGC itself means User Generated Content but in this context assume that such terms as "UGC ID" are specific to Valve's system. UGC IDs are found in various places in the API and Steam including decal attributes on TF2 items.

Practically speaking the purpose of *ugc_file* is similar to that of *steam.user.vanity_url*. Namely to convert an arbitrary ID into something useful like a direct URL.

class steam.remote_storage.**ugc_file**(*appid*, *ugcid64*, ***kwargs*)
Resolves a UGC file ID into usable metadata.

Fetches UGC file metadata for the given UGC and app ID.

```
>>> ugc = steam.remote_storage.ugc_file(440, 650994986817657344)
>>> ugc.url
u'http://images.akamai.steamusercontent.com/ugc/650994986817657344/D2ADAD7F19BFA9A99BD2B8850CC31
```

Properties:

size
Size in bytes

filename
Local filename is what the user named it, not the URL

url
UGC link

class steam.remote_storage.**FileNotFoundError**

User

class steam.user.**vanity_url**(*vanity*, ***kwargs*)
Class for holding a vanity URL and its id64

```
>>> vanity_url = steam.user.vanity_url('http://steamcommunity.com/id/ondrowan')
>>> vanity_url.id64
76561198017493014
```

class steam.user.**profile**(*sid*, ***kwargs*)
Functions for reading user account data

```
>>> profile = steam.user.profile('76561198017493014')
>>> profile.persona
u'Lich Buchannon'
>>> profile.level
37
```

id64
Returns the 64 bit steam ID (use with other API requests)

id32
Returns the 32 bit steam ID

persona
Returns the user's persona (what you usually see in-game)

profile_url
Returns a URL to the user's Community profile page

vanity
Returns the user's vanity url if it exists, None otherwise

avatar_small

avatar_medium

avatar_large

status
Returns the user's status. 0: offline 1: online 2: busy 3: away 4: snooze 5: looking to trade 6: looking to play If player's profile is private, this will always be 0.

visibility
Returns the visibility setting of the profile. 1: private 2: friends only 3: public

configured
Returns true if the user has created a Community profile

last_online
Returns the last time the user was online as a localtime time.struct_time struct

comments_enabled
Returns true if the profile allows public comments

real_name
Returns the user's real name if it's set and public

primary_group
Returns the user's primary group ID if set.

creation_date
Returns the account creation date as a localtime time.struct_time struct if public

current_game
Returns a tuple of 3 elements (each of which may be None if not available): Current game app ID, server ip:port, misc. extra info (eg. game title)

location
Returns a tuple of 2 elements (each of which may be None if not available): State ISO code, country ISO code

lobbysteamid
Returns a lobbynumber as int from few Source games or 0 if not in lobby.

level
Returns the the user's profile level, note that this runs a separate request because the profile level data isn't in the standard player summary output even though it should be. Which is also why it's not implemented as a separate class. You won't need this output and not the profile output

classmethod from_def(*obj*)
Builds a profile object from a raw player summary object

current_game
Returns a tuple of 3 elements (each of which may be None if not available): Current game app ID, server ip:port, misc. extra info (eg. game title)

class steam.user.**profile_batch**(*sids*)

```
>>> profiles = steam.user.profile_batch(['76561198014028523', '76561198017493014'])
>>> for profile in profiles:
...     profile.persona
...
u'Lagg'
u'Lich Buchannon'
```

class steam.user.**bans**(*sid*, ***kwargs*)

```
>>> bans = steam.user.bans('76561197962899758')
>>> bans.vac
True
>>> bans.vac_count
1
>>> bans.days_unbanned
2708
```

id64

community
Community banned

vac
User is currently VAC banned

vac_count
Number of VAC bans on record

days_unbanned
Number of days since the last ban. Note that users without a ban on record will have this set to 0 so make sure to test bans.vac

economy
Economy ban status which is a string for whatever reason

game_count
Number of bans in games, this includes CS:GO Overwatch bans

classmethod from_def(*obj*)

class steam.user.**bans_batch**(*sids*)

```
>>> bans_batch = steam.user.bans_batch(['76561197962899758', '76561198017493014'])
>>> for bans in bans_batch:
...     '%s: %s' % (bans.id64, bans.vac)
...
'76561197962899758: True'
'76561198017493014: False'
```

class steam.user.**friend**(*friend_dict*)
Class used to store friend obtained from GetFriendList.

steamid
Returns the 64 bit Steam ID

relationship
Returns relationship qualifier

since
Returns date when relationship was created as a localtime time.struct_time

class steam.user.**friend_list**(*sid*, *relationship='all'*, ***kwargs*)
Creates an iterator of friend objects fetched from given user's Steam ID. Allows for filtering by specyfing relationship argument in constructor, but API seems to always return items with friend relationship. Possible filter values: all, friend.

```
>>> friend_list = steam.user.friend_list('76561198014028523')
>>> friend_list.count
146
>>> for friend in friend_list:
```

```
...     friend.steamid
...
76561197960299337
76561197960339433
(... and 144 more)
```

count
Returns number of friends

CHAPTER 5

Steam Inventory Manager

High level item manager which scrapes data from http://steamcommunity.com instead of Steam API.

class steam.sim.**inventory_context**(*user*, ***kwargs*)

Builds context data that is fetched from a user's inventory page

Fetches metadata of inventories for different games of given user:

```
>>> inventory_context = steam.sim.inventory_context('76561198017493014')
>>> inventory_context.apps
[u'570', u'753', u'251970', u'440', u'620']
>>> inventory_context.get(570)
{u'name': u'Dota 2', u'trade_permissions': u'FULL', u'rgContexts': ...}
```

This class also acts as an iterator of inventories:

```
>>> for game_inventory_ctx in inventory_context:
...     game_inventory_ctx['name']
...
u'Team Fortress 2'
u'Dota 2'
u'Portal 2'
u'Steam'
u'Sins of a Dark Age'
```

Properties:

ctx

apps

Returns a list of valid app IDs

get(*key*)

Returns context data for a given app, can be an ID or a case insensitive name

class steam.sim.**inventory**(*app*, *profile*, *schema=None*, *section=None*, *timeout=None*, *lang=None*)

Takes *steam.sim.inventory_context* and user id, and fetches data from given inventory:

```
>>> inventory = steam.sim.inventory(inventory_context.get(570), '76561198017493014')
>>> inventory.cells_total
650
```

This class also acts as an iterator yielding *steam.sim.item* objects:

```
>>> for item in inventory:
...     item.full_name
...
```

```
u'Rattlebite'
u'Heavenly Guardian Skirt'
u'Gloried Horn of Druud'
...
```

Properties:

cells_total
Returns the total amount of "cells" which in this case is just an amount of items

class steam.sim.**item**(*theitem*, *context*)
Subclass of *steam.items.item*. It is used as output from *steam.sim.inventory*.

On top of properties inherited from *steam.items.item*, these are available:

attributes
Returns a list of attributes

category
Returns the category name that the item is a member of

background_color
Returns the color associated with the item as a hex RGB tuple

name

custom_name

name_color
Returns the name color as an RGB tuple

full_name

hash_name
The URL-friendly identifier for the item. Generates its own approximation if one isn't available

tool_metadata

tags
A list of tags attached to the item if applicable, format is subject to change

tradable

craftable

quality
Can't really trust presence of a schema here, but there is an ID sometimes

quantity

attributes

position

schema_id
This *will* return none if there is no schema ID, since it's a valve specific concept for the most part

type

icon

image

id

slot_name

appid
Return the app ID that this item belongs to

class steam.sim.**item_attribute**(*attribute*)
Subclass of *steam.items.item_attribute*. It is used as output from *steam.sim.item.attributes()*.

On top of properties inherited from *steam.items.item_attribute()*, these are available:

value_type

description

description_color
Returns description color as an RGB tuple

type

value

CHAPTER 6

VDF serializer

VDF is format similar to JSON or YAML, used by Valve to store data. This module mimics built-in `json` module and provides functions for serialization and deserialization of `VDF` files.

`steam.vdf.`**`dump`**(*obj*, *stream*)

Serializes *obj* as VDF formatted stream to *stream* object, encoded as UTF-16 by default.

```
>>> with open('dump.vdf', 'w') as file:
...     vdf.dump({u"key": u"value", u"list": [1, 2, 3]}, file)

→ cat dump.vdf
"list"
{
"1" "1"
"2" "1"
"3" "1"
}

"key" "value"
```

`steam.vdf.`**`dumps`**(*obj*)

Serializes *obj* as VDF formatted string, encoded as UTF-16 by default.

```
>>> vdf_obj = vdf.dumps({"key": "value", "list": [1, 2, 3]})
>>> vdf_obj.decode('utf-16')
u'\n  "list"\n  {\n    "1" "1"\n    "2" "1"\n    "3" "1"\n  }\n\n  "key" "value"\n'
```

`steam.vdf.`**`load`**(*stream*)

Deserializes *stream* containing VDF document to Python object.

```
>>> with open('dump.vdf', 'r') as file:
...     vdf.load(file)
...
{u'list': {u'1': u'1', u'3': u'1', u'2': u'1'}, u'key': u'value'}
```

`steam.vdf.`**`loads`**(*string*)

Deserializes *string* containing VDF document to Python object.

```
>>> vdf.loads('"list" { "a" "1" "b" "2" "c" "3" }')
{u'list': {u'a': u'1', u'c': u'3', u'b': u'2'}}
```

A

account_info (steam.items.item_attribute attribute), 14
app_list (class in steam.apps), 10
appid (steam.sim.item attribute), 22
apps (steam.sim.inventory_context attribute), 21
asset_item (class in steam.items), 15
assets (class in steam.items), 14
attributes (steam.items.item attribute), 11, 22
attributes (steam.items.schema attribute), 10
attributes (steam.sim.item attribute), 22
available_styles (steam.items.item attribute), 13
avatar_large (steam.user.profile attribute), 16
avatar_medium (steam.user.profile attribute), 16
avatar_small (steam.user.profile attribute), 16

B

background_color (steam.sim.item attribute), 22
bans (class in steam.user), 17
bans_batch (class in steam.user), 18
base_price (steam.items.asset_item attribute), 15

C

capabilities (steam.items.item attribute), 13
category (steam.sim.item attribute), 22
cells_total (steam.items.inventory attribute), 14
cells_total (steam.sim.inventory attribute), 22
client_url (steam.items.schema attribute), 10
code (steam.loc.language attribute), 15
comments_enabled (steam.user.profile attribute), 17
community (steam.user.bans attribute), 18
configured (steam.user.profile attribute), 17
contents (steam.items.item attribute), 12
count (steam.user.friend_list attribute), 19
craft_class (steam.items.item attribute), 12
craft_material_type (steam.items.item attribute), 12
craftable (steam.items.item attribute), 12
craftable (steam.sim.item attribute), 22
creation_date (steam.user.profile attribute), 17
ctx (steam.sim.inventory_context attribute), 21
current_game (steam.user.profile attribute), 17
custom_description (steam.items.item attribute), 12
custom_name (steam.items.item attribute), 12
custom_name (steam.sim.item attribute), 22
cvar_class (steam.items.item attribute), 12
cvar_class (steam.items.item_attribute attribute), 13

D

days_unbanned (steam.user.bans attribute), 18
description (steam.items.item attribute), 12
description (steam.items.item_attribute attribute), 13
description (steam.sim.item_attribute attribute), 23
description_color (steam.sim.item_attribute attribute), 23
dump() (in module steam.vdf), 25
dumps() (in module steam.vdf), 25

E

economy (steam.user.bans attribute), 18
equipable_classes (steam.items.item attribute), 11
equipped (steam.items.item attribute), 11

F

filename (steam.remote_storage.ugc_file attribute), 16
FileNotFoundError (class in steam.remote_storage), 16
formatted_description (steam.items.item_attribute attribute), 13
formatted_value (steam.items.item_attribute attribute), 13
friend (class in steam.user), 18
friend_list (class in steam.user), 18
from_def() (steam.user.bans class method), 18
from_def() (steam.user.profile class method), 17
full_name (steam.items.item attribute), 12
full_name (steam.sim.item attribute), 22

G

game_count (steam.user.bans attribute), 18
get() (steam.sim.inventory_context method), 21

H

hash_name (steam.sim.item attribute), 22
hidden (steam.items.item_attribute attribute), 13

I

icon (steam.items.item attribute), 11
icon (steam.sim.item attribute), 22
id (steam.items.item attribute), 12
id (steam.items.item_attribute attribute), 13
id (steam.sim.item attribute), 22
id32 (steam.user.profile attribute), 16
id64 (steam.user.bans attribute), 18
id64 (steam.user.profile attribute), 16
image (steam.items.item attribute), 12
image (steam.sim.item attribute), 22
inventory (class in steam.items), 14
inventory (class in steam.sim), 21
inventory_context (class in steam.sim), 21
inventory_token (steam.items.item attribute), 11
item (class in steam.items), 11
item (class in steam.sim), 22
item_attribute (class in steam.items), 13
item_attribute (class in steam.sim), 23

K

kill_eaters (steam.items.item attribute), 12
kill_ranks (steam.items.schema attribute), 11
kill_types (steam.items.schema attribute), 11

L

language (class in steam.loc), 15
language (steam.items.schema attribute), 10
LanguageUnsupportedError (class in steam.loc), 15
last_online (steam.user.profile attribute), 17
level (steam.items.item attribute), 12
level (steam.user.profile attribute), 17
load() (in module steam.vdf), 25
loads() (in module steam.vdf), 25
lobbysteamid (steam.user.profile attribute), 17
location (steam.user.profile attribute), 17

M

min_level (steam.items.item attribute), 12

N

name (steam.items.asset_item attribute), 15
name (steam.items.item attribute), 11
name (steam.items.item_attribute attribute), 13
name (steam.loc.language attribute), 15
name (steam.sim.item attribute), 22
name_color (steam.sim.item attribute), 22

O

origin (steam.items.item attribute), 13
original_id (steam.items.item attribute), 12
origins (steam.items.schema attribute), 10

P

particle_systems (steam.items.schema attribute), 10
persona (steam.user.profile attribute), 16
position (steam.items.item attribute), 11
position (steam.sim.item attribute), 22
price (steam.items.asset_item attribute), 15
primary_group (steam.user.profile attribute), 17
profile (class in steam.user), 16
profile_batch (class in steam.user), 17
profile_url (steam.user.profile attribute), 16

Q

qualities (steam.items.schema attribute), 10
quality (steam.items.item attribute), 11
quality (steam.sim.item attribute), 22
quantity (steam.items.item attribute), 12
quantity (steam.sim.item attribute), 22

R

rank (steam.items.item attribute), 12
real_name (steam.user.profile attribute), 17
relationship (steam.user.friend attribute), 18

S

schema (class in steam.items), 10
schema_id (steam.items.item attribute), 11
schema_id (steam.sim.item attribute), 22
since (steam.user.friend attribute), 18
size (steam.remote_storage.ugc_file attribute), 16
slot_name (steam.items.item attribute), 12
slot_name (steam.sim.item attribute), 22
status (steam.user.profile attribute), 17
steamid (steam.user.friend attribute), 18
style (steam.items.item attribute), 13

T

tags (steam.items.asset_item attribute), 15
tags (steam.items.assets attribute), 14
tags (steam.sim.item attribute), 22
tool_metadata (steam.items.item attribute), 13
tool_metadata (steam.sim.item attribute), 22
tradable (steam.items.item attribute), 12
tradable (steam.sim.item attribute), 22
type (steam.items.item attribute), 11
type (steam.items.item_attribute attribute), 13
type (steam.sim.item attribute), 22
type (steam.sim.item_attribute attribute), 23

U

ugc_file (class in steam.remote_storage), 16
url (steam.remote_storage.ugc_file attribute), 16

V

vac (steam.user.bans attribute), 18
vac_count (steam.user.bans attribute), 18
value (steam.items.item_attribute attribute), 13
value (steam.sim.item_attribute attribute), 23
value_float (steam.items.item_attribute attribute), 13
value_int (steam.items.item_attribute attribute), 13
value_type (steam.items.item_attribute attribute), 13
value_type (steam.sim.item_attribute attribute), 23
vanity (steam.user.profile attribute), 16
vanity_url (class in steam.user), 16
visibility (steam.user.profile attribute), 17