librarian Documentation

Release 0.3.0

Taylor "Nekroze" Lawson

October 28, 2013

CONTENTS

Python advanced card game library.

CHAPTER

ONE

FEATURES

- Sqlite based card storage database
- Easy de/re-serialization of card objects
- Complex filtering methods for card databases

1.1 Contents:

1.1.1 Installation

At the command line either via easy_install or pip:

```
$ easy_install librarian
$ pip install librarian
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv librarian
$ pip install librarian
```

1.1.2 librarian

librarian Package

librarian Package

card Module

Generic Card Class.

class librarian.card.**Card**(*code=None*, *name=None*, *loaddict=None*)

Bases: object

The card stores general information about the card.

- code: the unique identifier for this card.
- name: name of this card to be displayed.
- abilities: dict of phase ids containing a list of action descriptors.

- attributes: list of special details this card has.
- info: dict of any information you would like.

Card can be saved to, and loaded from, a string. Call `str()` on a Card instance or `.save_string()` on the instance. This will return a string that when evaluated using `eval()` can be unpacked into the Card constructor re-create that card. For example. `original = Card(1, 'cool card') savestring = str(card) loaded = Card(*eval(savestring)) assert loaded == original`

add_ability(*phase*, *ability*)
Add the given ability to this Card under the given phase. Returns the length of the abilities for the given phase after the addition.

add_attribute(*attribute*)
Add the given attribute to this Card. Returns the length of attributes after addition.

get_abilities(*phase*)
Returns an ability list for the given phase ID.

get_info(*key*)
Return a value in the info for this card with the given key.

has_attribute(*attribute*)
Return true if this card contains the given attribute.

is_valid()
Returns True if code is not 0 and self.name is not ''.

load(*carddict*)
Takes a carddict as produced by `Card.save` and sets this card instances information to the previously saved cards information.

save()
Converts the Card as is into a dictionary capable of reconstructing the card with `Card.load` or serialized to a string for storage.

set_info(*key*, *value*, *append=True*)
Set any special info you wish to the given key. Each info is stored in a list and will be appended to rather then overriden unless append is False.

deck Module

Generic Card Class.

class librarian.deck.**Deck**(*library=None*, *cards=None*)
Bases: object

A collection of possibly recuring cards stored as codes.

contains_card(*code*)
Returns true if the given code is currently stored in this deck.

contains_info(*key*, *value*)
Returns how many cards in the deck have the specified value under the specified key in their info data.

This method requires a library to be stored in the deck instance and will return *None* if there is no library.

contians_attribute(*attribute*)
Returns how many cards in the deck have the specified attribute.

This method requires a library to be stored in the deck instance and will return *None* if there is no library.

get_card(*index=-1*, *cache=True*, *remove=True*)
Retrieve a card any number of cards from the top. Returns a `Card` object loaded from a library if one is specified otherwise just it will simply return its code.

If *index* is not set then the top card will be retrieved.

If cache is set to True (the default) it will tell the library to cache the returned card for faster look-ups in the future.

If remove is true then the card will be removed from the deck before returning it.

move_top_cards(*other*, *number=1*)
Move the top *number* of cards to the top of some *other* deck.

By default only one card will be moved if *number* is not specified.

remaining()
Returns the number of remaining cards in the deck.

shuffle()
Sort the cards in the deck into a random order..

top_cards(*number=1*, *cache=True*, *remove=True*)
Retrieve the top number of cards as `Librarian.Card` objects in a list in order of top to bottom most card. Uses the decks `.get_card` and passes along the cache and remove arguments.

`library` Module

The Library class, an sqlite database of cards.

class `librarian.library.`**Library**(*dbname*, *cachelimit=100*, *cardclass=<class 'librarian.card.Card'>*)
Bases: `object`

Library wraps an sqlite3 database that stores serialized cards.

Library also allows load and save hooks that allow a list of function to be called on each string as it is saved and loaded.

The `Library` constructor can take a `cardclass` argument which defaults to `librarian.card.Card` and is used to construct a card object when loading. A `cardclass` should be a subclass of `librarian.card.Card` and be able to take the original `carddict` constructor argument alone along with providing the original or equal `Card.load` and `Card.save` methods.

cache_card(*card*)
Cache the card for faster future lookups. Removes the oldest card when the card cache stores more cards then this libraries cache limit.

cached(*code*)
Return True if there is a card for the given code in the cache.

connection()
Connect to the underlying database and return the connection.

create_db()
Create the CARDS table in the sqlite3 database.

filter_search(*code=None*, *name=None*, *abilities=None*, *attributes=None*, *info=None*)
Return a list of codes and names pertaining to cards that have the given information values stored.

Can take a code integer, name string, abilities dict {phase: ability list/"*"}, attributes list, info dict {key, value list/"*"}.

In the above argument examples "*" is a string that may be passed instead of a list as the dict value to match anything that stores that key.

load_card(*code*, *cache=True*)
Load a card with the given code from the database. This calls each save event hook on the save string before commiting it to the database.

Will cache each resulting card for faster future lookups with this method while respecting the libraries cache limit. However only if the cache argument is True.

Will return None if the card could not be loaded.

retrieve_all()
A generator that iterates over each card in the library database.

This is best used in for loops as it will only load a card from the library as needed rather then all at once.

save_card(*card*, *cache=False*)
Save the given card to the database. This calls each save event hook on the save string before commiting it to the database.

librarian.library.**Where_filter_gen**(**data*)
Generate an sqlite "LIKE" filter generator based on the given data. This functions arguments should be a N length series of field and data tuples.

1.1.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/Nekroze/librarian/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

librarian could always use more documentation, whether as part of the official librarian docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Nekroze/librarian/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *librarian* for local development.

1. Fork the *librarian* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/librarian.git
   ```

3. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

Now you can make your changes locally.

4. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

   ```
   $ tox
   ```

To get tox, just pip install it.

5. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

6. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/Nekroze/librarian under pull requests for active pull requests or run the `tox` command and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test test/test_librarian.py
```

1.1.4 Credits

Development Lead

- Taylor "Nekroze" Lawson <nekroze@eturnilnetwork.com>

Contributors

None yet. Why not be the first?

1.1.5 History

0.3.0 (09/02/2013)

- First re-release on PyPI.

1.2 Feedback

If you have any suggestions or questions about **librarian** feel free to email me at nekroze@eturnilnetwork.com.

If you encounter any errors or problems with **librarian**, please let me know! Open an Issue at the GitHub http://github.com/Nekroze/librarian main repository.

PYTHON MODULE INDEX

l