ObservableList Documentation

Release 3

niccokunzmann

Jul 22, 2016

Contents

Contents:

CHAPTER 1

ObservableList Installation Instructions

1.1 Package installation from Pypi

The ObservableList library requires Python 3. It can be installed form the Python Package Index.

1.1.1 Windows

Install it with a specific python version under windows:

```
py -3 -m pip --no-cache-dir install --upgrade ObservableList
```

Test the installed version:

```
py -3 -m pytest --pyargs ObservableList
```

1.1.2 Linux

To install the version from the python package index, you can use your terminal and execute this under Linux:

```
sudo python3 -m pip --no-cache-dir install --upgrade ObservableList
```

test the installed version:

```
python3 -m pytest --pyargs ObservableList
```

1.2 Installation from Repository

You can setup the development version under Windows and Linux.

1.2.1 Linux

If you wish to get latest source version running, you can check out the repository and install it manually.

```
git clone https://github.com/niccokunzmann/ObservableList.git
cd ObservableList
sudo python3 -m pip install --upgrade pip
sudo python3 -m pip install -r requirements.txt
sudo python3 -m pip install -r test-requirements.txt
py.test
```

To also make it importable for other libraries, you can link it into the site-packages folder this way:

```
sudo python3 setup.py link
```

1.2.2 Windows

Same as under *Linux* but you need to replace `sudo python3` with `py -3`. This also counts for the following documentation.

CHAPTER 2

Development Setup

Make sure that you have the *repository installed*.

2.1 Install Requirements

To install all requirements for the development setup, execute

```
pip install --upgrade -r requirements.txt -r test-requirements.txt -r dev-
↪requirements.txt
```

2.2 Sphinx Documentation Setup

Sphinx was setup using the tutorial from readthedocs. It should be already setup if you completed *the previous step*.

Further reading:

- domains

With Notepad++ under Windows, you can run the make_html.bat file in the `docs` directory to create the documentation and show undocumented code.

2.3 Code Climate

To install the code climate command line interface (cli), read about it in their github repository You need docker to be installed. Under Linux you can execute this in the Terminal to install docker:

```
wget -qO- https://get.docker.com/ | sh
sudo usermod -aG docker $USER
```

Then, log in and out. Then, you can install the command line interface:

```
wget -qO- https://github.com/codeclimate/codeclimate/archive/master.tar.gz | tar xvz
cd codeclimate-* && sudo make install
```

Then, go to the ObservableList repository and analyze it.

```
codeclimate analyze
```

2.4 Version Pinning

We use version pinning, described in this blog post (outdated). Also read the current version for how to set up.

After installation you can run

```
pip install -r requirements.in -r test-requirements.in -r dev-requirements.in
pip-compile --output-file requirements.txt requirements.in
pip-compile --output-file test-requirements.txt test-requirements.in
pip-compile --output-file dev-requirements.txt dev-requirements.in
pip-sync requirements.txt dev-requirements.txt test-requirements.txt
pip install --upgrade -r requirements.txt -r test-requirements.txt -r dev-
↪requirements.txt
```

`pip-sync` uninstalls every package you do not need and writes the fix package versions to the requirements files.

2.5 Continuous Integration to Pypi

Before you put something on Pypi, ensure the following:

1. The version is in the master branch on github.
2. The tests run by travis-ci run successfully.

Pypi is automatically deployed by travis. See here. To upload new versions, tag them with git and push them.

```
setup.py tag_and_deploy
```

The tag shows up as a travis build. If the build succeeds, it is automatically deployed to Pypi.

2.6 Manual Upload to the Python Package Index

However, here you can see how to upload this package manually.

2.6.1 Version

Throughout this chapter, `<new_version>` refers to a a string of the form `[0-9]+\.[0-9]+\.[0-9]+[ab]?` or `<MAYOR>.<MINOR>.<STEP>[<MATURITY>]` where `<MAYOR>`, `<MINOR>` and, `<STEP>` represent numbers and `<MATURITY>` can be a letter to indicate how mature the release is.

1. Create a new branch for the version.

```
git checkout -b <new_version>
```

2. Increase the `__version__` in __init__.py
   - no letter at the end means release
   - `b` in the end means Beta
   - `a` in the end means Alpha
3. Commit and upload this version.

```
git add ObservableList/__init__.py
git commit -m "version <new_version>"
git push origin <new_version>
```

4. Create a pull-request.
5. Wait for travis-ci to pass the tests.
6. Merge the pull-request.
7. Checkout the master branch and pull the changes from the *commit*.

```
git checkout master
git pull
```

8. Tag the version at the master branch with a `v` in the beginning and push it to github.

```
git tag v<new_version>
git push origin v<new_version>
```

9. *Upload* the code to Pypi.

2.6.2 Upload

First ensure all tests are running:

```
setup.py pep8
```

From docs.python.org:

```
setup.py sdist bdist_wininst upload register
```

2.7 Classifiers

You can find all Pypi classifiers here.

CHAPTER 3

The `ObservableList` Module Reference

3.1 `ObservableList` Module

This module contains the ObservableList.

This list works like a normal list but additionally observers can be registered that are notified, whenever the list is changed.

class `ObservableList.`**`ObservableList`** (*iterable=()*)

Bases: `list`

The observable list behaves like a list but changes can be observed.

See the Observer Pattern for more understanding.

The methods "clear" and "copy" are not available in Python 2.

`__delitem__` (*index_or_slice*)

Delete self[key].

`__iadd__` (*other*)

Implement self+=value.

`__imul__` (*multiplier*)

Implement self*=value.

`__init__` (*iterable=()*)

Initialize self. See help(type(self)) for accurate signature.

`__setitem__` (*index_or_slice*, *value*)

Set self[key] to value.

`__weakref__`

list of weak references to the object (if defined)

`append` (*object*) → None – append object to end

`clear` ( ) → None – remove all items from L

`extend` (*iterable*) → None – extend list by appending elements from the iterable

`insert` (*index*, *item*)

L.insert(index, object) – insert object before index

`notify_observers` (*change*)

Notify the observers about the change.

pop ([*index*]) → item – remove and return item at index (default last).
Raises IndexError if list is empty or index is out of range.

register_observer (*observer*)
Register an observer.

Parameters **observer** – callable that takes a *Change* as first argument

remove (*value*) → None – remove first occurrence of value.
Raises ValueError if the value is not present.

class ObservableList.**Change** (*list_*, *slice_*)
Bases: object

The base class for changes.

__init__ (*list_*, *slice_*)
Initialize the change.

Parameters

- **list_** (*list*) – the list that changed
- **slice_** (*slice*) – the slice of the `list_` to change

__repr__ ( )
The change as string.

__weakref__
list of weak references to the object (if defined)

adds ( )
Whether the change adds values.

Return type bool

changed_object
The object that was changed.

Returns the object that was changed

elements
The elements affected by this change.

Return type list

items ( )
Return an iterable over pairs of index and value.

length
The number of elements changed.

Return type int

```
assert change.length == len(change.indices)
assert change.length == len(change.elements)
```

range
The indices of the changed objects.

Return type *range*

removes ( )
Whether the change removes values.

Return type bool

start
The start index of the change.

Return type int

step
The step size of the change.

Return type int

stop
The stop index of the change.

Return type int

Note: As with lists, the element at the stop index is excluded from the change.

class `ObservableList.`**`AddChange`**`(`*list_*, *slice_*`)`
Bases: *`ObservableList.Change`*

A change that adds elements.

adds()
Whether the change adds values (True).

Return type bool

Returns `True`

class `ObservableList.`**`RemoveChange`**`(`*list_*, *slice_*`)`
Bases: *`ObservableList.Change`*

A change that removes elements.

removes()
Whether the change removes values (True).

Return type bool

Returns `True`

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index