xdmenu Documentation

Release

Charles Bouchard-Légaré

Jun 09, 2017

Contents

Extensible wrapper for dmenu.

> *dmenu is a dynamic menu for X, originally designed for dwm. It manages large numbers of user-defined menu items efficiently.*

- Source code on GitHub
- Latest documentation

xdmenu is free software and licensed under the GNU Lesser General Public License v3.

CHAPTER 1

Features

- Many options available in patches built in
- Additional options can be added
- Easy to extend for other tools such as Rofi

CHAPTER 2

Credits

This package was created with Cookiecutter and the cblegare/pythontemplate project template.

Contents:

Installation

xdmenu uses and needs an implementation of *dmenu*. This means a command line program that reads lines from *stdin*, presents these lines to the user as a menu and prints the chosen lines to *stdout*.

dmenu *dmenu is a dynamic menu for X, originally designed for dwm. It manages large numbers of user-defined menu items efficiently.*

dmenu2 *dmenu2 is the fork of original dmenu - an efficient dynamic menu for X, patched with XFT, quiet, x & y, token, fuzzy matching, follow focus, tab nav, filter.*

Added option to set screen on which dmenu apperars, as long as opacity, window class and window name. Also allows to dim screen with selected color and opacity while dmenu2 is running.

Added underline color and height. (options -uc and -uh)

Rofi *Rofi, like dmenu, will provide the user with a textual list of options where one or more can be selected. This can either be, running an application, selecting a window or options provided by an external script.*

Stable release

To install *xdmenu*, run this command in your terminal:

```
$ pip install xdmenu
```

This is the preferred method to install *xdmenu*, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for *xdmenu* can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/cblegare/xdmenu
```

Or download the tarball:

```
$ curl -OL https://github.com/cblegare/xdmenu/tarball/master
```

Once you have a copy of the source, you can install it with:

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

Usage

xdmenu is a wrapper API for dmenu. The original use case of *xdmenu* was to ease the integration of *dmenu* with Qtile, a window manager written in Python.

The simplest possible usage of this wrapper is through the *xdmenu.dmenu()* function. Here is an example usage:

```
>>> from xdmenu import dmenu
>>> dmenu(['foo', 'bar'])  # shows a menu window with choices on one line
['bar']                    # the user picked 'bar'
>>> dmenu(['foo', 'bar'], lines=2)  # shows a menu window with two lines
['foo']                             # the user picked 'foo'
```

xdmenu.**dmenu**(*choices*, *dmenu=None*, ***kwargs*)

Run *dmenu* with configuration provided in `**kwargs`.

Parameters

- **choices** (*list*) – Choices to put in menu
- **dmenu** (xdmenu.BaseMenu) – A *xdmenu.BaseMenu* instance to use. If not provided, a default one will be created.
- ****kwargs** – Any of the supported argument added via *xdmenu.BaseMenu.add_arg()*.

Returns All the choices made by the user.

Return type list

See also:

xdmenu.BaseMenu.run()

The *xdmenu* package also provides the *xdmenu.Dmenu* class. This class can be provided with default configuration values to customize the behavior of *dmenu*.

class xdmenu.**Dmenu**(*proc_runner=None*, ***kwargs*)

An extensible dmenu wrapper that already supports all usual arguments.

Parameters

- **dmenu** (*str*) – See *xdmenu.BaseMenu()*
- **proc_runner** (*Callable[[list, list], str]*) – See *xdmenu.BaseMenu()*

- **bottom** (*bool*) – dmenu appears at the bottom of the screen. Equivalent for the `-b` command line option of dmenu.
- **grab** (*bool*) – dmenu grabs the keyboard before reading stdin. This is faster, but will lock up X until stdin reaches end-of-file. Equivalent for the `-f` command line option of dmenu.
- **insensitive** (*bool*) – dmenu matches menu items case insensitively. Equivalent for the `-i` command line option of dmenu.
- **lines** (*int*) – dmenu lists items vertically, with the given number of lines. Equivalent for the `-l` command line option of dmenu.
- **monitor** (*int*) – dmenu is displayed on the monitor number supplied. Monitor numbers are starting from 0. Equivalent for the `-m` command line option of dmenu.
- **prompt** (*str*) – defines the prompt to be displayed to the left of the input field. Equivalent for the `-p` command line option of dmenu.
- **font** (*str*) – defines the font or font set used. Equivalent for the `-fn` command line option of dmenu.
- **normal_bg_color** (*str*) – defines the normal background color. #RGB, #RRGGBB, and X color names are supported. Equivalent for the `-nb` command line option of dmenu.
- **normal_fg_color** (*str*) – defines the normal foreground color. Equivalent for the `-nf` command line option of dmenu.
- **selected_bg_color** (*str*) – defines the selected background color. Equivalent for the `-sb` command line option of dmenu.
- **selected_fg_color** (*str*) – defines the selected foreground color. Equivalent for the `-sf` command line option of dmenu.
- **windowid** (*str*) – embed into windowid.

Run *dmenu* using *xdmenu.BaseMenu.run()* which all child class should have.

BaseMenu.**run**(*choices*, ***kwargs*)

Parameters

- **choices** (*list*) – Choices to put in menu
- ****kwargs** – See *xdmenu.BaseMenu.configure()*, except that values are no kept for a later call to dmenu

Examples

```
>>> # We mock the _run_dmenu_process function for this example
>>> # to be runnable even if dmenu is not installed
>>> # The mock mimics a user choosing the first choice
>>> m = Dmenu(proc_runner=_mock_dmenu_process)
>>> m.run(['foo', 'bar'])
['foo']
```

Returns All the choices made by the user. In order to have multiple results, a custom build of *dmenu* may be required since the original version may not support selecting many items.

Return type list

If you only want to get the command line arguments, simply use *xdmenu.BaseMenu.make_cmd()*

BaseMenu.**make_cmd**(***kwargs*)

Build the list of command line arguments to dmenu.

Parameters ****kwargs** – See *xdmenu.BaseMenu.configure()*, except that values are no kept for a later call to dmenu

Returns

List of command parts ready to sead to `subprocess.Popen`

Return type list

Examples

```
>>> menu = Dmenu()
>>> menu.make_cmd()
['dmenu']
>>> menu.make_cmd(bottom=True)
['dmenu', '-b']
>>> menu.make_cmd(lines=2, prompt='-> ',)
['dmenu', '-l', '2', '-p', '-> ']
```

Since *xdmenu* is intended to be extensible, you can add supported options using *xdmenu.BaseMenu.add_arg()*

BaseMenu.**add_arg**(*name*, *converter*, *default=None*)

Extend this wrapper by registering a new dmenu argument.

You can also use this to change the behavior of existing arguments.

Parameters

- **name** (*str*) – The name of the supported keyword argument for this wrapper.
- **converter** (*Callable[[Any], Iterable]*) – A function that converts the configured value to a list of command line arguments to dmenu.
- **default** (*Optional[Any]*) – The default configured value.

Examples

Let's wrap the usage of a *-foo* argument that a dmenu fork could possibly support.

```
>>> def to_bottom(arg):
...     return ['-foo'] if arg else []
>>> menu = Dmenu()
>>> menu.add_arg('foo', to_bottom, default=False)
>>> menu.make_cmd()
['dmenu']
>>> menu.make_cmd(foo=True)
['dmenu', '-foo']
```

xdmenu also provides a wrapper for dmenu2. See *xdmenu.Dmenu2*.

CHAPTER 3

Project Information

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/cblegare/xdmenu/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about the build an version of *dmenu* that you use.
- 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" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

xdmenu could always use more documentation, whether as part of the official *xdmenu* 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/cblegare/xdmenu/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 *xdmenu* for local development.

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

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

3. Install your local copy into a virtualenv. Assuming you have Python 3.5 installed, this is how you set up your fork for local development:

```
$ python3 -m venv xdmenu
$ cd xdmenu/
$ bin/pip install --editable . # or bin/python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. 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
```

7. 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 2.7 and up. Check https://travis-ci.org/cblegare/xdmenu/pull_requests and make sure that the tests pass for all supported Python versions.

Also, have a look at the full-fledged *Setup Script*!

Thanks :)

History

1.0.1 (2017-06-01)

- Fixed: Infinite recursive loop when using *Dmenu2* constructor

1.0.0 (2017-06-01)

- First release on PyPI.

License

```
GNU LESSER GENERAL PUBLIC LICENSE

                    Version 3, 29 June 2007

    xdmenu
    Copyright (C) 2017  Charles Bouchard-Légaré

    xdmenu is free software: you can redistribute it and/or modify
    it under the terms of the GNU Lesser General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    xdmenu is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU Lesser General Public License for more details.

    You should have received a copy of the GNU Lesser General Public License
    along with xdmenu.  If not, see <http://www.gnu.org/licenses/>.
```

Credits

Contributors

- Charles Bouchard-Légaré <cblegare.atl@ntis.ca>

CHAPTER 4

Development resources

Setup Script

The *setup.py* file is a swiss knife for various tasks.

Start by creating a virtual python environment:

```
$ python -m venv .
```

You now can use this isolated clean python environment:

```
$ bin/python --version
Python 3.5.2
```

You may also activate it for the current shell. POSIX shells would use:

```
$ . bin/activate
```

running tests

We use py.test for running tests because it is amazing. Run it by invoking the simple *test* alias of *setup.py*:

```
$ bin/python setup.py test
```

This will also check codestyle and test coverage.

checking code style

We use flake8 for enforcing coding standards. Run it by invoking the simple *lint* alias of *setup.py*:

```
$ bin/python setup.py lint
```

building source distirbutions

Standard *sdist* is supported:

```
$ bin/python setup.py sdist
```

building binary distributions

Use the wheel distribution standard:

```
$ bin/python setup.py bdist_wheel
```

building html documentation

Use *setup.py* to build the documentation:

```
$ bin/python setup.py docs
```

A make implementation is not required on any platform, thanks to the *setup.Documentation* class.

class setup.**Documentation**(*dist*, ***kw*)
Make the documentation (without the *Make* program).

Note: This command will not allow any warning from Sphinx, treating them as errors.

Construct the command for dist, updating vars(self) with any keyword parameters.

cleaning your workspace

We also included a custom command which you can invoke through *setup.py*:

```
$ bin/python setup.py clean
```

The *setup.Clean* command is set to clean the following file patterns:

class setup.**Clean**(*dist*, ***kw*)
Custom clean command to tidy up the project.

Construct the command for dist, updating vars(self) with any keyword parameters.

default_patterns = ['build', 'dist', '*.egg-info', '*.egg', '*.pyc', '*.pyo', '*~', '__pycache__', '.tox', '.coverage', 'html

Automated tests

The *tests* package provides automated testing for *'xdmenu'*.

Tests are known to assess software behavior and find bugs. They are also used as part of the code's documentation, as a design tool or for preventing regressions.

See also:

- http://stackoverflow.com/questions/4904096/whats-the-difference-between-unit-functional-acceptance-and-integration-test

- http://stackoverflow.com/questions/520064/what-is-unit-test-integration-test-smoke-test-regression-test

Unit tests

Exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Unit tests should not

- call out into (non-trivial) collaborators,
- access the network,
- hit a database,
- use the file system or
- spin up a thread.

Most of the unit tests can be found directory in the code documentation and are run using doctest. When they cannot be simple or extensible enough with impeding readability, they should be written in the `tests.unit` package.

Integration tests

Verify the communication paths and interactions between components to detect interface defects.

The line between unit and integration tests may become blurry. When in doubt, you are most certainly thinking integration tests. Write those in the `tests.integration` package.

Functional tests

Functional tests check a particular feature for correctness by comparing the results for a given input against the specification. They are often used as an executable definition of a user story. Write those in the `tests.functional` package.

Regression tests

A test that was written when a bug was found (and then fixed). It ensures that this specific bug will not occur again. The full name is *non-regression test*. It can also be a test made prior to changing an application to make sure the application provides the same outcome. Put these in the `tests.regression` package.

All Automated tests

tests package

Subpackages

tests.functional package

Functional tests for xdmenu.

tests.integration package

Integration tests for xdmenu.

tests.regression package

Non-regression tests for xdmenu.

tests.unit package

Unit tests for xdmenu.

Most of unit tests are doctests directly next to the production code.

API documentation

xdmenu package

Package main definition.

xdmenu.**dmenu**(*choices*, *dmenu=None*, ***kwargs*)

Run *dmenu* with configuration provided in `**kwargs`.

Parameters

- **choices** (*list*) – Choices to put in menu
- **dmenu** (xdmenu.BaseMenu) – A *xdmenu.BaseMenu* instance to use. If not provided, a default one will be created.
- ****kwargs** – Any of the supported argument added via *xdmenu.BaseMenu.add_arg()*.

Returns All the choices made by the user.

Return type list

See also:

xdmenu.BaseMenu.run()

class xdmenu.**BaseMenu**(*dmenu=None*, *proc_runner=None*, ***kwargs*)

Bases: object

An extensible dmenu wrapper.

Parameters

- **dmenu** (*str*) – dmenu executable to use.
- **proc_runner** (*Callable[[list, list], str]*) – a function that calls dmenu as a subprocess and returns the output. This defaults to a simple call to subprocess.Popen.
- ****kwargs** – See *xdmenu.BaseMenu.configure()*

add_arg(*name*, *converter*, *default=None*)
Extend this wrapper by registering a new dmenu argument.

You can also use this to change the behavior of existing arguments.

Parameters

- **name** (*str*) – The name of the supported keyword argument for this wrapper.
- **converter** (*Callable[[Any], Iterable]*) – A function that converts the configured value to a list of command line arguments to dmenu.
- **default** (*Optional[Any]*) – The default configured value.

Examples

Let's wrap the usage of a *-foo* argument that a dmenu fork could possibly support.

```
>>> def to_bottom(arg):
...     return ['-foo'] if arg else []
>>> menu = Dmenu()
>>> menu.add_arg('foo', to_bottom, default=False)
>>> menu.make_cmd()
['dmenu']
>>> menu.make_cmd(foo=True)
['dmenu', '-foo']
```

configure(***kwargs*)
Set a value to any of the supported argument added.

See also:

xdmenu.BaseMenu.add_arg().

Parameters ****kwargs** – Keywords are mapped to the name of the argument, and the value is kept for a future call to dmenu.

make_cmd(***kwargs*)
Build the list of command line arguments to dmenu.

Parameters ****kwargs** – See *xdmenu.BaseMenu.configure()*, except that values are no kept for a later call to dmenu

Returns

List of command parts ready to sead to `subprocess.Popen`

Return type list

Examples

```
>>> menu = Dmenu()
>>> menu.make_cmd()
['dmenu']
>>> menu.make_cmd(bottom=True)
['dmenu', '-b']
>>> menu.make_cmd(lines=2, prompt='-> ',)
['dmenu', '-l', '2', '-p', '-> ']
```

run (*choices*, ***kwargs*)

Parameters

- **choices** (*list*) – Choices to put in menu
- ****kwargs** – See *xdmenu.BaseMenu.configure()*, except that values are no kept for a later call to dmenu

Examples

```
>>> # We mock the _run_dmenu_process function for this example
>>> # to be runnable even if dmenu is not installed
>>> # The mock mimics a user choosing the first choice
>>> m = Dmenu(proc_runner=_mock_dmenu_process)
>>> m.run(['foo', 'bar'])
['foo']
```

Returns All the choices made by the user. In order to have multiple results, a custom build of *dmenu* may be required since the original version may not support selecting many items.

Return type list

version (*dmenu=None*)

Return dmenu version string.

Parameters **dmenu** (*str*) – dmenu executable to use. Defaults to the one configured in *self*.

Returns The configured dmenu's version string

Return type str

class xdmenu.**Dmenu** (*proc_runner=None*, ***kwargs*)

Bases: *xdmenu.BaseMenu*

An extensible dmenu wrapper that already supports all usual arguments.

Parameters

- **dmenu** (*str*) – See *xdmenu.BaseMenu()*
- **proc_runner** (*Callable[[list, list], str]*) – See *xdmenu.BaseMenu()*
- **bottom** (*bool*) – dmenu appears at the bottom of the screen. Equivalent for the `-b` command line option of dmenu.
- **grab** (*bool*) – dmenu grabs the keyboard before reading stdin. This is faster, but will lock up X until stdin reaches end-of-file. Equivalent for the `-f` command line option of dmenu.
- **insensitive** (*bool*) – dmenu matches menu items case insensitively. Equivalent for the `-i` command line option of dmenu.
- **lines** (*int*) – dmenu lists items vertically, with the given number of lines. Equivalent for the `-l` command line option of dmenu.
- **monitor** (*int*) – dmenu is displayed on the monitor number supplied. Monitor numbers are starting from 0. Equivalent for the `-m` command line option of dmenu.
- **prompt** (*str*) – defines the prompt to be displayed to the left of the input field. Equivalent for the `-p` command line option of dmenu.

- **font** (*str*) – defines the font or font set used. Equivalent for the `-fn` command line option of dmenu.
- **normal_bg_color** (*str*) – defines the normal background color. #RGB, #RRGGBB, and X color names are supported. Equivalent for the `-nb` command line option of dmenu.
- **normal_fg_color** (*str*) – defines the normal foreground color. Equivalent for the `-nf` command line option of dmenu.
- **selected_bg_color** (*str*) – defines the selected background color. Equivalent for the `-sb` command line option of dmenu.
- **selected_fg_color** (*str*) – defines the selected foreground color. Equivalent for the `-sf` command line option of dmenu.
- **windowid** (*str*) – embed into windowid.

class xdmenu.**Dmenu2**(*proc_runner=None*, ***kwargs*)

Bases: *xdmenu.Dmenu*

A wrapper for dmenu2.

This wrapper also supports all of *xdmenu.Dmenu* arguments in addition to the ones below.

Parameters

- **dmenu** (*str*) – See *xdmenu.BaseMenu()*
- **proc_runner** (*Callable[[list, list], str]*) – See *xdmenu.BaseMenu()*
- **filter** (*bool*) – activates filter mode. All matching items currently shown in the list will be selected, starting with the item that is highlighted and wrapping around to the beginning of the list. Equivalent for the `-r` command line option of dmenu2.
- **fuzzy** (*bool*) – dmenu uses fuzzy matching. It matches items that have all characters entered, in sequence they are entered, but there may be any number of characters between matched characters. For example it takes `txt` makes it to `*t*x*t` glob pattern and checks if it matches. Equivalent for the `-z` command line option of dmenu2.
- **token** (*bool*) – dmenu uses space-separated tokens to match menu items. Using this overrides *fuzzy* option. Equivalent for the `-t` command line option of dmenu2.
- **mask** (*bool*) – dmenu masks input with asterisk characters (*). Equivalent for the `-mask` command line option of dmenu2.
- **screen** (*int*) – dmenu apears on the specified screen number. Number given corresponds to screen number in X configuration. Equivalent for the `-s` command line option of dmenu2.
- **window_name** (*str*) – defines window name for dmenu. Defaults to “dmenu”. Equivalent for the `-name` command line option of dmenu2.
- **window_class** (*str*) – defines window class for dmenu. Defaults to “Dmenu”. Equivalent for the `-class` command line option of dmenu2.
- **opacity** (*float*) – defines window opacity for dmenu. Defaults to 1.0. Equivalent for the `-o` command line option of dmenu2.
- **dim** (*float*) – enables screen dimming when dmenu appers. Takes dim opacity as argument. Equivalent for the `-dim` command line option of dmenu2.
- **dim_color** (*str*) – defines color of screen dimming. Active only when -dim in effect. Defautls to black (#000000). Equivalent for the `-dc` command line option of dmenu2.
- **height** (*int*) – defines the height of the bar in pixels. Equivalent for the `-h` command line option of dmenu2.

- **xoffset** (*int*) – defines the offset from the left border of the screen. Equivalent for the `-x` command line option of dmenu2.
- **yoffset** (*int*) – defines the offset from the top border of the screen. Equivalent for the `-y` command line option of dmenu2.
- **width** (*int*) – defines the desired menu window width. Equivalent for the `-w` command line option of dmenu2.

exception xdmenu.**DmenuError**(*args*, *stderr*)

Bases: Exception

Something went wrong with dmenu.

exception xdmenu.**DmenuUsageError**(*args*, *stderr*)

Bases: *xdmenu.DmenuError*

Some arguments to dmenu where invalid.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

x

Index