whiptail

Release 0.4.0

Use whiptail to display dialog boxes from Python scripts.

Dominic Davis-Foster

Aug 31, 2022

Contents

`whiptail` is a library that will let you present a variety of questions or display messages using dialog boxes from a Python script.

Currently, these types of dialog boxes are implemented:

- yes/no box
- menu box
- input box
- message box
- text box
- info box
- checklist box
- radiolist box
- gauge box
- password box

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install whiptail-dialogs --user
```

1.2 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/whiptail@master --user
```

You must also have the `whiptail` package installed on your system.

On Debian and derivatives this can be installed with:

```
$ apt-get install whiptail
```

1.3 Example Usage

Listing 1: `example.py`

```
# this package
from whiptail import Whiptail

w = Whiptail(title="This is the title", backtitle="This is the backtitle")

prompt = w.inputbox("Enter some text:")[0]
print(f"You entered: '{prompt}'!")

prompt_default = w.inputbox("Enter some text:", "Some Text ;)")[0]
print(f"You entered: '{prompt_default}'!")

prompt_password = w.inputbox("Enter a (pretend) password:", password=True)[0]
print(f"Your password is: '{prompt_password}'!")

msgbox = w.msgbox("This is a msgbox!")  # type: ignore
print(f"msgbox doesn't return anything, see: {msgbox}")

menu = w.menu("This is a menu.", ["Option 1", "Option 2", "Option 3", "Option 4"])[0]
print(f"You selected '{menu}'")

menu_descriptions = w.menu(
                "This is a menu with descriptions.",
                [("Option 1", "Does Something"), ("Option 2", "Does Something Else")],
                )[0]
print(f"You selected '{menu_descriptions}'")
```

(continues on next page)

(continued from previous page)

```
radiolist = w.radiolist("Choose One", ["Spam, spam, spam, spam", "Egg", "Chips"])[0]
print(f"You selected: '{radiolist}'!")

checklist = w.checklist("Choose Multiple", ["Spam, spam, spam, spam", "Egg", "Chips
↪"])[0]
checklist_str = "' and '".join(checklist)
print(f"You selected: '{checklist_str}'!")

textbox = w.textbox(__file__)
print(textbox)
```

1.4 API Reference

Use whiptail to display dialog boxes from Python scripts.

Classes:

Response(returncode, value)	Namedtuple to store the returncode and value returned by a whiptail dialog.
Whiptail([title, backtitle, height, width, ...])	Display dialog boxes in the terminal from Python scripts.

namedtuple Response (*returncode*, *value*)

Bases: NamedTuple

Namedtuple to store the returncode and value returned by a whiptail dialog.

Fields

0) **returncode** (int) – The returncode.

1) **value** (str) – The value returned from the dialog.

Return values are as follows:

- 0: The Yes or OK button was pressed.
- 1: The No or Cancel button was pressed.
- 255: The user pressed the ESC key, or an error occurred.

static __new__ (*cls*, *returncode*, *value*)

Create a new instance of *Response*.

Parameters

- **returncode** (int) – The returncode.
- **value** (AnyStr) – The value returned from the dialog.

class Whiptail (*title=''*, *backtitle=''*, *height=None*, *width=None*, *auto_exit=False*)

Bases: object

Display dialog boxes in the terminal from Python scripts.

Parameters

- **title** (str) – The text to show at the top of the dialog. Default ''.
- **backtitle** (str) – The text to show on the top left of the background. Default ''.
- **height** (Optional[int]) – The height of the dialog. Default is 2-5 characters shorter than the terminal window
- **width** (Optional[int]) – The height of the dialog. Default is approx. 10 characters narrower than the terminal window
- **auto_exit** (bool) – Whether to call sys.exit() if the user selects cancel in a dialog. Default False.

Methods:

calc_height(msg)	Calculate the height of the dialog box based on the message.
checklist([msg, items, prefix])	A checklist box is similar to a menu box in that there are multiple entries presented in the form of a menu.
inputbox(msg[, default, password])	An input box is useful when you want to ask questions that require the user to input a string as the answer.
menu([msg, items, prefix])	As its name suggests, a menu box is a dialog box that can be used to present a list of choices in the form of a menu for the user to choose.
msgbox(msg)	A message box is very similar to a yes/no box.
radiolist([msg, items, prefix])	A radiolist box is similar to a menu box.
run(control, msg[, extra_args, ...])	Display a control.
showlist(control, msg, items, prefix)	Helper function to display radio- and check-lists.
textbox(path)	A text box lets you display the contents of a text file in a dialog box.
yesno(msg[, default])	Display a yes/no dialog box.

calc_height(*msg*)

Calculate the height of the dialog box based on the message.

Parameters **msg** (str) – The message to display in the dialog box

Return type List[str]

checklist(*msg=''*, *items=()*, *prefix='-'*)

A checklist box is similar to a menu box in that there are multiple entries presented in the form of a menu.

You can select and deselect items using the SPACE key. The initial on/off state of each entry is specified by status.

Parameters

- **msg** (str) – The message to display in the dialog box. Default ''.
- **items** (Union[Sequence[str], Sequence[Iterable[str]]]) – A sequence of items to display in the checklist. Default ().
- **prefix** (str) – Default ' - '.

Return type Tuple[List[str], int]

Returns A list of the tag strings of those entries that are turned on, and the return code

inputbox(*msg*, *default=''*, *password=False*)

An input box is useful when you want to ask questions that require the user to input a string as the answer. If `default` is supplied it is used to initialize the input string. When inputting the string, the `BACKSPACE` key can be used to correct typing errors. If the input string is longer than the width of the dialog box, the input field will be scrolled.

If `password` is `True`, the text the user enters is not displayed. This is useful when prompting for passwords or other sensitive information. Be aware that if anything is passed in "init", it will be visible in the system's process table to casual snoopers. Also, it is very confusing to the user to provide them with a default password they cannot see. For these reasons, using "init" is highly discouraged.

Parameters

- **msg** (`str`) – The message to display in the dialog box
- **default** (`str`) – A default value for the text. Default `''`.
- **password** (`bool`) – Whether the text being entered is a password, and should be replaced by `*`. Default `False`. Default `False`.

Return type `Tuple[str, int]`

Returns The value entered by the user, and the return code

menu(*msg=''*, *items=()*, *prefix='- '*)

As its name suggests, a menu box is a dialog box that can be used to present a list of choices in the form of a menu for the user to choose.

Each menu entry consists of a tag string and an item string. The tag gives the entry a name to distinguish it from the other entries in the menu. The item is a short description of the option that the entry represents. The user can move between the menu entries by pressing the `UP`/`DOWN` keys, the first letter of the tag as a hot-key. There are menu-height entries displayed in the menu at one time, but the menu will be scrolled if there are more entries than that.

Parameters

- **msg** (`str`) – The message to display in the dialog box. Default `''`.
- **items** (`Union[Sequence[str], Sequence[Iterable[str]]]`) – A sequence of items to display in the menu. Default `()`.
- **prefix** (`str`) – Default `' - '`.

Return type `Tuple[str, int]`

Returns The tag of the selected menu item, and the return code.

msgbox(*msg*)

A message box is very similar to a yes/no box.

The only difference between a message box and a yes/no box is that a message box has only a single `OK` button.

You can use this dialog box to display any message you like. After reading the message the user can press the ENTER key so that whiptail will exit and the calling script can continue its operation.

Parameters **msg** (`str`) – The message to display in the dialog box

radiolist(*msg=''*, *items=()*, *prefix=' - '*)
A radiolist box is similar to a menu box.

The only difference is that you can indicate which entry is currently selected, by setting its status to on.

Parameters

- **msg** (str) – The message to display in the dialog box. Default `''`.
- **items** (Union[Sequence[str], Sequence[Iterable[str]]]) – A sequence of items to display in the radiolist. Default `()`.
- **prefix** (str) – Default `' - '`.

Return type Tuple[List[str], int]

Returns A list of the tags strings that were selected, and the return code.

run(*control*, *msg*, *extra_args=()*, *extra_values=()*, *exit_on=(1, 255)*)
Display a control.

Parameters

- **control** (str) – The name of the control to run. One of `'yesno'`, `'msgbox'`, `'infobox'`, `'inputbox'`, `'passwordbox'`, `'textbox'`, `'menu'`, `'checklist'`, `'radiolist'` or `'gauge'`
- **msg** (str) – The message to display in the dialog box
- **extra_args** (Sequence[str]) – A sequence of extra arguments to pass to the control. Default `()`.
- **extra_values** (Sequence[str]) – A sequence of extra values to pass to the control. Default `()`.
- **exit_on** (Sequence[int]) – A sequence of return codes that will cause program execution to stop if `Whiptail.auto_exit` is `True`. Default `(1, 255)`.

Return type *Response*

Returns The response returned by whiptail

showlist(*control*, *msg*, *items*, *prefix*)
Helper function to display radio- and check-lists.

Parameters

- **control** (Literal['checklist', 'radiolist']) – The name of the control to run. Either `'checklist'` or `'radiolist'`.
- **msg** (str) – The message to display in the dialog box/
- **items** (Union[Sequence[str], Sequence[Iterable[str]]]) – A sequence of items to display in the list/
- **prefix** (str)

Return type Tuple[List[str], int]

Returns A list of the tags strings that were selected, and the return code/

textbox(*path*)
A text box lets you display the contents of a text file in a dialog box. It is like a simple text file viewer. The user can move through the file by using the UP/DOWN, PGUP/PGDN and HOME/END keys available on

most keyboards. If the lines are too long to be displayed in the box, the LEFT/RIGHT keys can be used to scroll the text region horizontally. For more convenience, forward and backward searching functions are also provided.

Parameters **path** (Union[str, Path, PathLike]) – The file to display the contents of

Return type int

Returns The return code

yesno(*msg*, *default='yes'*)

Display a yes/no dialog box.

The string specified by msg is displayed inside the dialog box. If this string is too long to be fit in one line, it will be automatically divided into multiple lines at appropriate places. The text string may also contain the newline character \n to control line breaking explicitly.

This dialog box is useful for asking questions that require the user to answer either yes or no. The dialog box has a Yes button and a No button, in which the user can switch between by pressing the TAB key.

Parameters

- **msg** (str) – The message to display in the dialog box
- **default** (str) – The default button to select, either 'yes' or 'no'. Default 'yes'.

Return type bool

Returns True if the user selected yes. False otherwise.

1.5 Contributing

whiptail uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install pre-commit with pip and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

1.5.1 Coding style

formate is used for code formatting.

It can be run manually via pre-commit:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

1.5.2 Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

1.5.3 Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

1.5.4 Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

1.6 Downloading source code

The `whiptail` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/whiptail

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/whiptail
```

```
Cloning into 'whiptail'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

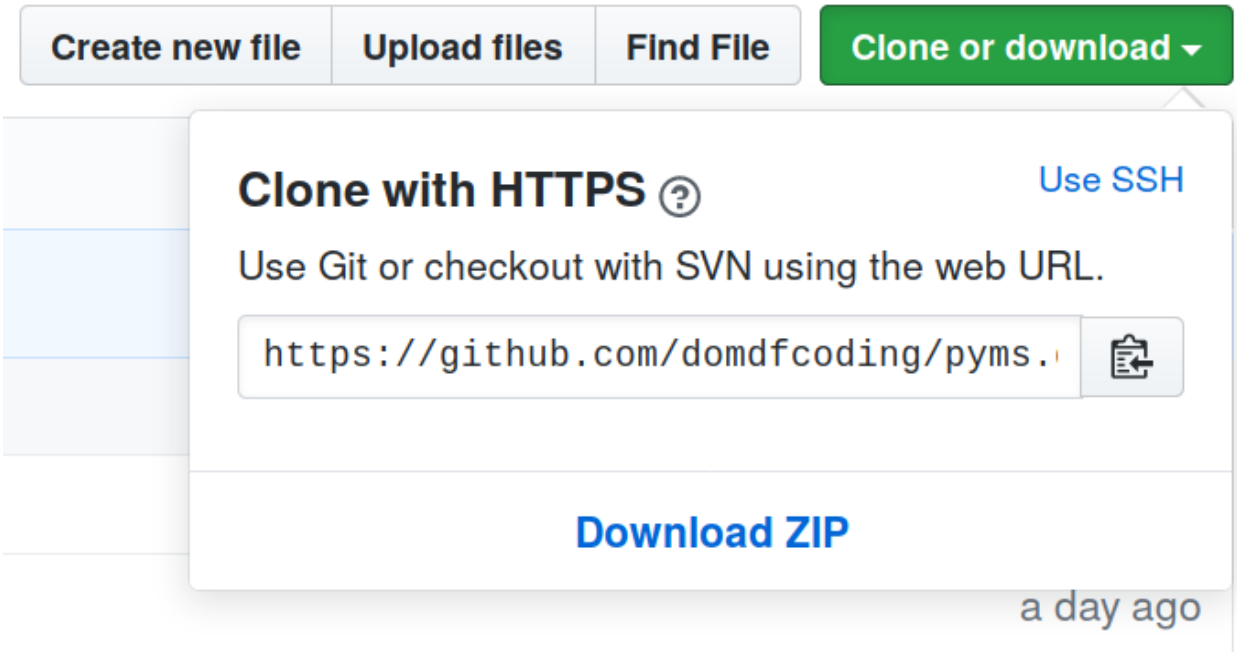

Fig. 1: Downloading a 'zip' file of the source code

1.6.1 Building from source

The recommended way to build `whiptail` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

1.7 License

`whiptail` is licensed under the BSD 3-Clause "New" or "Revised" License

---

A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the copyright holder or its contributors to promote derived products without written consent.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

```
BSD 3-Clause License

Copyright (c) 2020 Dominic Davis-Foster <dominic@davis-foster.co.uk>
Copyright (c) 2013 Marwan Alsabbagh and contributors.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
   contributors may be used to endorse or promote products derived from
   this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Python Module Index

W

Index

Symbols

B

C

I

M

P

R

S

T

V

W

Y