easygui$_{qt}$Documentation

Release 0.9.2

André Roberge

Apr 05, 2017

Contents

Contents:

CHAPTER 1

EasyGUI_Qt

Inspired by EasyGUI, designed for PyQt

EasyGUI_Qt is a module for simple and easy GUI programming in Python.

EasyGUI_Qt was inspired by EasyGUI created by Stephen Ferg and is based on Tkinter. By contrast, EasyGUI_Qt is based on PyQt which is not included in the standard Python distribution - but is included in some other distributions like Continuum Analytics' Anaconda.

- Free software: BSD license
- Documentation: https://easygui_qt.readthedocs.org.

Python version

Officially, this is a project that targets only Python 3. However, I have now decided to attempt to provide some support for Python 2. Other than some unicode issues, all widgets should work with Python 2.

Design philosophy

Like the original EasyGUI, EasyGUI_Qt seeks to provide simple GUI widgets that can be called in a procedural program. EasyGUI_Qt is NOT event-driven: all GUI interactions are invoked by simple function calls.

The archetype is `get_string(message)` which pops a box whose purpose is exactly the same as Python's `input(prompt)`, that is, present the user with a question/prompt, have the user enter an answer, and return the provided answer as a string. Thus `easygui_qt.get_string()` can be used as a drop-in replacement for `input()`. Similarly, instead of using a `print()` function to display a message, `show_message()` is used which pops a message window; however, note that unlike `print`, `show_message` interrupts the flow of the program and require some interaction from the user for the program to continue.

Unlike the original EasyGUI, which sometimes used cryptic names like `msgbox` or `ynbox`, EasyGUI_Qt attempts to use descriptive names which follow PEP8 convention. Thus, instead of `msgbox`, it uses `show_message`; instead of `ynbox`, it has `get_yes_or_no`. Most function names start with either `get_`, `show_` or `set_`.

EasyGUI_QT is based on PyQt; it leverages the available dialogs that come with PyQt whenever possible. This makes it possible to have automatic translation of some GUI elements (such as text on standard buttons) provided the locale is set correctly and that the local distribution of PyQt includes the appropriate translation: when EasyGUI_Qt runs, it scans the standard PyQt location for translation files and note which ones are present and can be used when the locale is set.

An attempt is made at avoiding duplication of essentially identical functionality. Thus, multiple selections from a list of choices is done only one way: by using a dialog where choices appear as labels in text and not labels on buttons.

Roadmap

See https://github.com/aroberge/easygui_qt/issues/13 and feel free to add comments.

Similar projects

The following is an incomplete lists of a few cross-platform projects that share some similarity with EasyGUI_Qt, but use back-ends other than PyQt

- easygui: the original; tkinter back-end
- anygui: multiple back-ends; well known but no longer supported
- psidialogs: multiple back-ends supported - possibly the most complete project from that point of view.
- python-dialog: dialog/Xdialog/gdialog back-end

There are quite a few lesser known projects but none that seem to be actively supported. If you are aware of other projects that should be mentioned, do not hesitate to contact me and let me know.

CHAPTER 2

Installation

Prerequisite:

PyQt4

At the command line:

```
$ easy_install easygui_qt
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv easygui_qt
$ pip install easygui_qt
```

CHAPTER 3

Usage

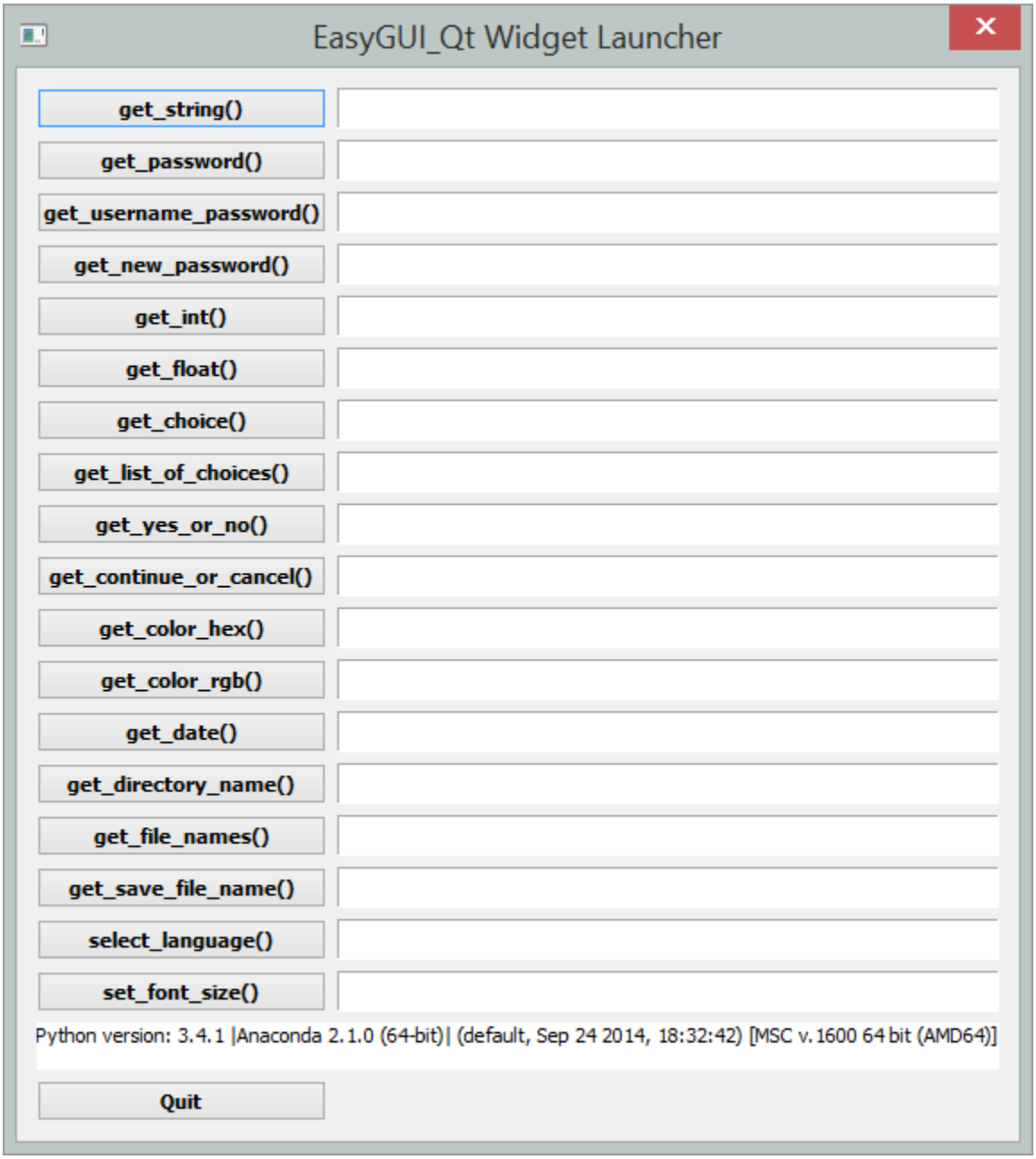
To use easygui_qt in a project:

```
import easygui_qt
```

Demos

There are currently two demos. The first one is a "launcher" which allows one to see each existing "widget" in action with its default values. You can run this demo as follows:

```
>>> from easygui_qt.demos import launcher
>>> launcher.main()
```



There is also another type of demo, whose intention is more to show how EasyGUI_Qt might be used in a "real-life" situation.

```
>>> from easygui_qt.demos.guessing_game import guessing_game
>>> guessing_game()
```

CHAPTER 4

Naming convention

Warning: The naming convention is currently used as a guide helping to finalize the API. Not all widgets listed here are implemented yet, or may be implemented using slightly different names.

In order to make its use more intuitive, EasyGUI_Qt uses a consistent naming convention.

All instructions meant to display information to a user without getting a response back start with `show`. The functions available are:

- `show()`
- `show_code()`
- `show_file()`
- `show_story()`

Note that a detailed description of all of these is given on the next page.

When a response is expected from the user, the prefixed used is `get_`. Thus we have, in alphabetical order:

- `get_abort()`
- `get_button()`
- `get_choice()`
- `get_color_hex()`
- `get_color_rgb()`
- `get_continue_or_cancel()`
- `get_date()`
- `get_directory_name()`
- `get_file_names()`
- `get_float()`

- `get_int()`
- `get_integer()`
- `get_language()`
- `get_list_of_choices()`
- `get_many_strings()`
- `get_new_password()`
- `get_password()`
- `get_save_file_name()`
- `get_string()`
- `get_username_password()`
- `get_yes_or_no()`

One exception to the above is the special widget used to handle exceptions, appropriately called:

- `handle_exception()`

Functions with no corresponding graphical component can be used to set some global parameters; they are prefixed by `set_`:

- `set_font_size()`
- `set_language()`

Finally, when writing code, instead of using Python's `help()` function, one can simply use following function which will open the API page on the ReadTheDocs website:

- `find_help()`

Specifying arguments

Arguments are all keyword based arguments. However, in order to enable simplified entry, they are generally listed in a consistent way.

The first argument is `message`: this is the text that appears in the window itself and is usually the most important information that is conveyed to the user.

CHAPTER 5

EasyGUI_Qt API

EasyGUI_Qt: procedural gui based on PyQt

EasyGUI_Qt is inspired by EasyGUI and contains a number of different basic graphical user interface components

`easygui_qt.easygui_qt.`**`get_choice`**(*message='Select one item'*, *title='Title'*, *choices=None*)

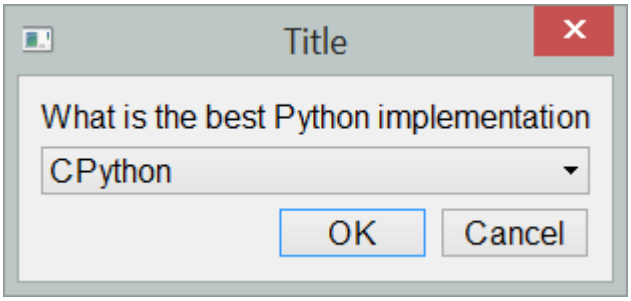
Simple dialog to ask a user to select an item within a drop-down list

Parameters

- **`message`** – Message displayed to the user, inviting a response
- **`title`** – Window title
- **`choices`** – iterable (list or tuple) containing the names of the items that can be selected.

Returns a string, or `None` if "cancel" is clicked or window is closed.

```
>>> import easygui_qt as easy
>>> choices = ["CPython", "Pypy", "Jython", "IronPython"]
>>> reply = easy.get_choice("What is the best Python implementation",
...                         choices=choices)
```



`easygui_qt.easygui_qt.`**`get_list_of_choices`**(*title='Title'*, *choices=None*)

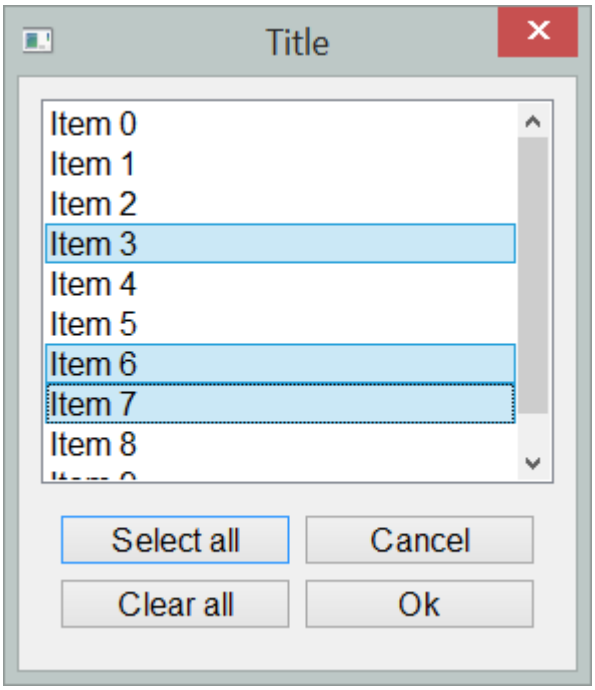
Show a list of possible choices to be selected.

Parameters

- **title** – Window title
- **choices** – iterable (list, tuple, ...) containing the choices as strings

Returns a list of selected items, otherwise an empty list.

```
>>> import easygui_qt as easy
>>> choices = easy.get_list_of_choices()
```



easygui_qt.easygui_qt.**get_float**(*message='Choose a number'*, *title='Title'*, *default_value=0.0*, *min_=-10000*, *max_=10000*, *decimals=3*)

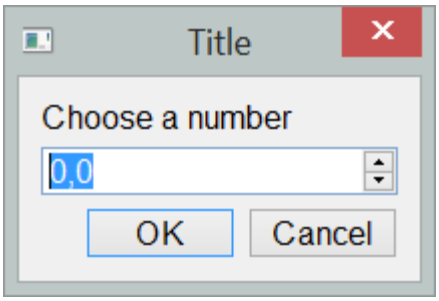
Simple dialog to ask a user to select a floating point number within a certain range and a maximum precision.

Parameters

- **message** – Message displayed to the user, inviting a response
- **title** – Window title
- **default_value** – Default value for value appearing in the text box; set to the closest of `min_` or `max_` if outside of allowed range.
- **min** – Minimum value allowed
- **max** – Maximum value allowed
- **decimals** – Indicate the maximum decimal precision allowed

Returns a floating-point number, or `None` if "cancel" is clicked or window is closed.

```
>>> import easygui_qt as easy
>>> number = easy.get_float()
```



Note: depending on the locale of the operating system where this is used, instead of a period being used for indicating the decimals, a comma may appear instead; this is the case for the French version of Windows for example. Therefore, entry of floating point values in this situation will require the use of a comma instead of a period. However, the internal representation will still be the same, and the number passed to Python will be using the familar notation.

easygui_qt.easygui_qt.**get_int**(*message='Choose a number'*, *title='Title'*, *default_value=1*, *min_=0*, *max_=100*, *step=1*)

Simple dialog to ask a user to select an integer within a certain range.

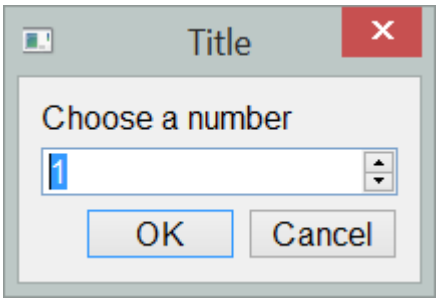
Note: **get_int()** and **get_integer()** are identical.

Parameters

- **message** – Message displayed to the user, inviting a response
- **title** – Window title
- **default_value** – Default value for integer appearing in the text box; set to the closest of `min_` or `max_` if outside of allowed range.
- **min** – Minimum integer value allowed
- **max** – Maximum integer value allowed
- **step** – Indicate the change in integer value when clicking on arrows on the right hand side

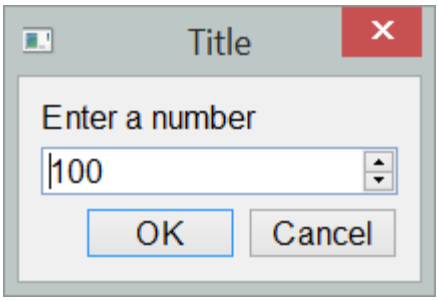
Returns an integer, or `None` if "cancel" is clicked or window is closed.

```
>>> import easygui_qt as easy
>>> number = easy.get_int()
```



If `default_value` is larger than `max_`, it is set to `max_`; if it is smaller than `min_`, it is set to `min_`.

```
>>> number = easy.get_integer("Enter a number", default_value=125)
```



easygui_qt.easygui_qt.**get_integer**(*message='Choose a number'*, *title='Title'*, *default_value=1*, *min_=0*, *max_=100*, *step=1*)

Simple dialog to ask a user to select an integer within a certain range.

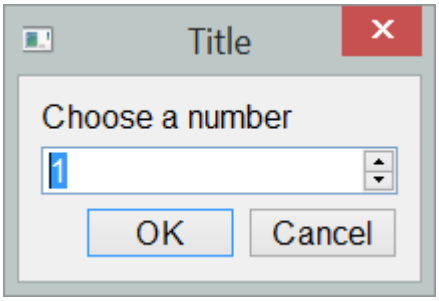
Note: **get_int()** and **get_integer()** are identical.

Parameters

- **message** – Message displayed to the user, inviting a response
- **title** – Window title
- **default_value** – Default value for integer appearing in the text box; set to the closest of `min_` or `max_` if outside of allowed range.
- **min** – Minimum integer value allowed
- **max** – Maximum integer value allowed
- **step** – Indicate the change in integer value when clicking on arrows on the right hand side

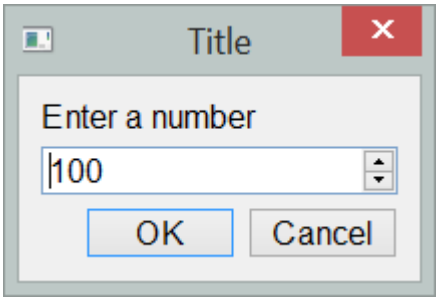
Returns an integer, or `None` if "cancel" is clicked or window is closed.

```
>>> import easygui_qt as easy
>>> number = easy.get_int()
```



If `default_value` is larger than `max_`, it is set to `max_`; if it is smaller than `min_`, it is set to `min_`.

```
>>> number = easy.get_integer("Enter a number", default_value=125)
```



easygui_qt.easygui_qt.**get_string**(*message='Enter your response'*, *title='Title'*, *default_response=''*)

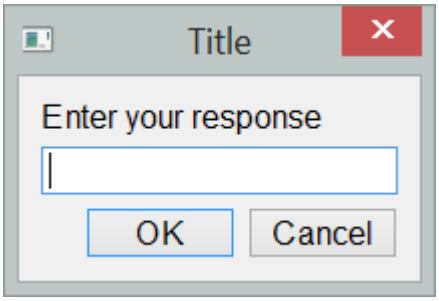
Simple text input box. Used to query the user and get a string back.

Parameters

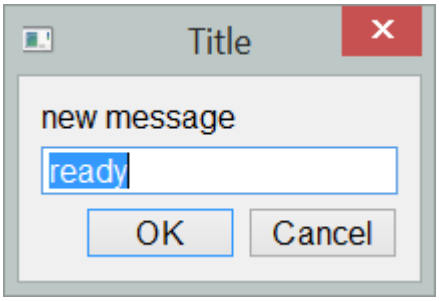
- **message** – Message displayed to the user, inviting a response
- **title** – Window title
- **default_response** – default response appearing in the text box

Returns a string, or `None` if "cancel" is clicked or window is closed.

```
>>> import easygui_qt as easy
>>> reply = easy.get_string()
```




```
>>> reply = easy.get_string("new message", default_response="ready")
```



easygui_qt.easygui_qt.**get_many_strings**(*title='Title'*, *labels=None*, *masks=None*)

Multiple strings input

Parameters

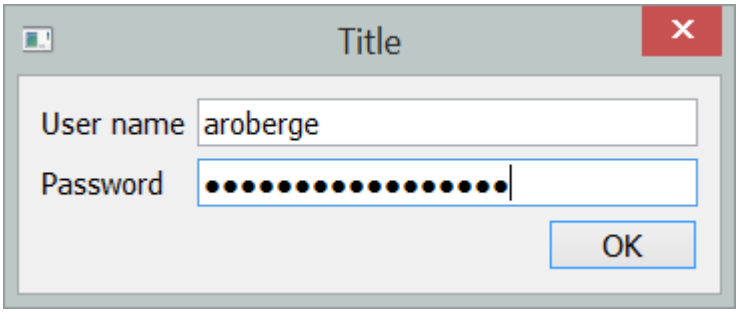
- **title** – Window title
- **labels** – an iterable containing the labels for to use for the entries
- **masks** – optional parameter.

Returns An ordered dict containing the labels as keys, and the input from the user (empty string by default) as value

The parameter `masks` if set must be an iterable of the same length as `choices` and contain either True or False as entries indicating if the entry of the text is masked or not. For example, one could ask for a username and password using get_many_strings as follows [note that get_username_password exists and automatically takes care of specifying the masks and is a better choice for this use case.]

```
>>> import easygui_qt as easy
>>> labels = ["User name", 'Password']
```

```
>>> masks = [False, True]
>>> reply = easy.get_many_strings(labels=labels, masks=masks)
>>> reply
OrderedDict([('User name', 'aroberge'), ('Password', 'not a good password')])
```



easygui_qt.easygui_qt.**get_password**(*message='Enter your password'*, *title='Title'*)

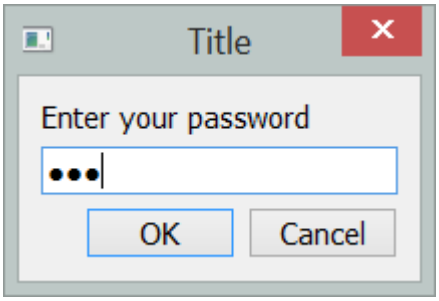
Simple password input box. Used to query the user and get a string back.

Parameters

- **message** – Message displayed to the user, inviting a response
- **title** – Window title

Returns a string, or None if "cancel" is clicked or window is closed.

```
>>> import easygui_qt as easy
>>> password = easy.get_password()
```



easygui_qt.easygui_qt.**get_username_password**(*title='Title'*, *labels=None*)

User name and password input box.

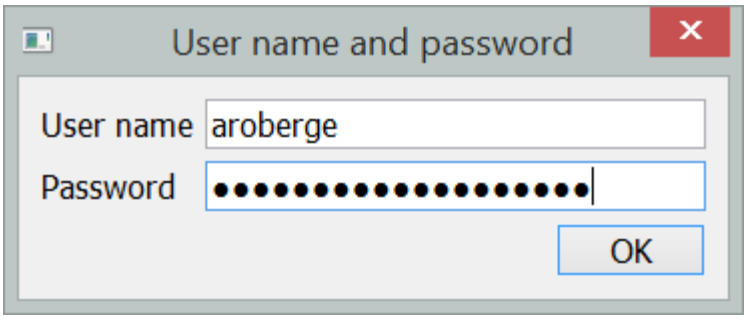
Parameters

- **title** – Window title
- **labels** – an iterable containing the labels for "user name" and "password"; if the value not specified, the default values will be used.

Returns An ordered dict containing the fields item as keys, and the input from the user (empty string by default) as value

Note: this function is a special case of get_many_strings where the required masks are provided automatically..

```
>>> import easygui_qt as easy
>>> reply = easy.get_username_password()
>>> reply
OrderedDict([('User name', 'aroberge'), ('Password', 'not a good password')])
```



easygui_qt.easygui_qt.**get_new_password**(*title='Title'*, *labels=None*)

Change password input box.

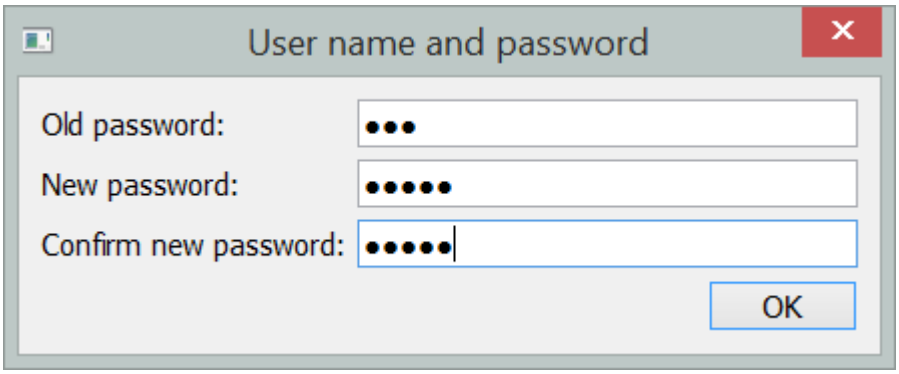
Parameters

- **title** – Window title
- **labels** – an iterable containing the labels for "Old password" and "New password" and "Confirm new password". All three labels must be different strings as they are used as keys in a dict - however, they could differ only by a space.

Returns An ordered dict containing the fields item as keys, and the input from the user as values.

Note: this function is a special case of `get_many_strings` where the required masks are provided automatically..

```
>>> import easygui_qt as easy
>>> reply = easy.get_new_password()
```



easygui_qt.easygui_qt.**get_yes_or_no**(*message='Answer this question'*, *title='Title'*)

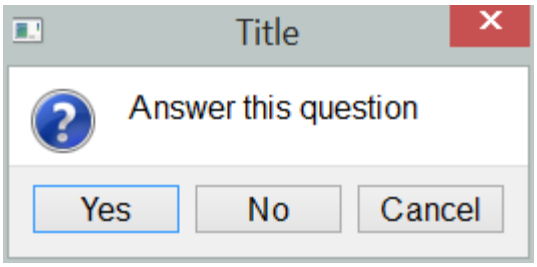
Simple yes or no question.

Parameters

- **question** – Question (string) asked
- **title** – Window title (string)

Returns `True` for "Yes", `False` for "No", and `None` for "Cancel".

```
>>> import easygui_qt as easy
>>> choice = easy.get_yes_or_no()
```



easygui_qt.easygui_qt.**get_continue_or_cancel**(*message='Processed will be cancelled!'*, *title='Title'*, *continue_button_text='Continue'*, *cancel_button_text='Cancel'*)

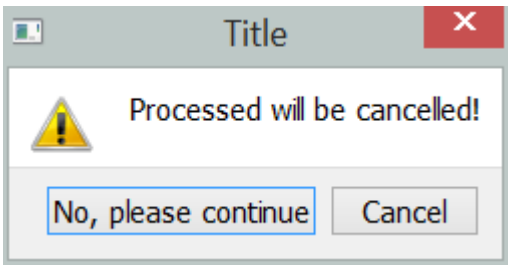
Continue or cancel question, shown as a warning (i.e. more urgent than simple message)

Parameters

- **question** – Question (string) asked
- **title** – Window title (string)
- **continue_button_text** – text to display on button
- **cancel_button_text** – text to display on button

Returns True for "Continue", False for "Cancel"

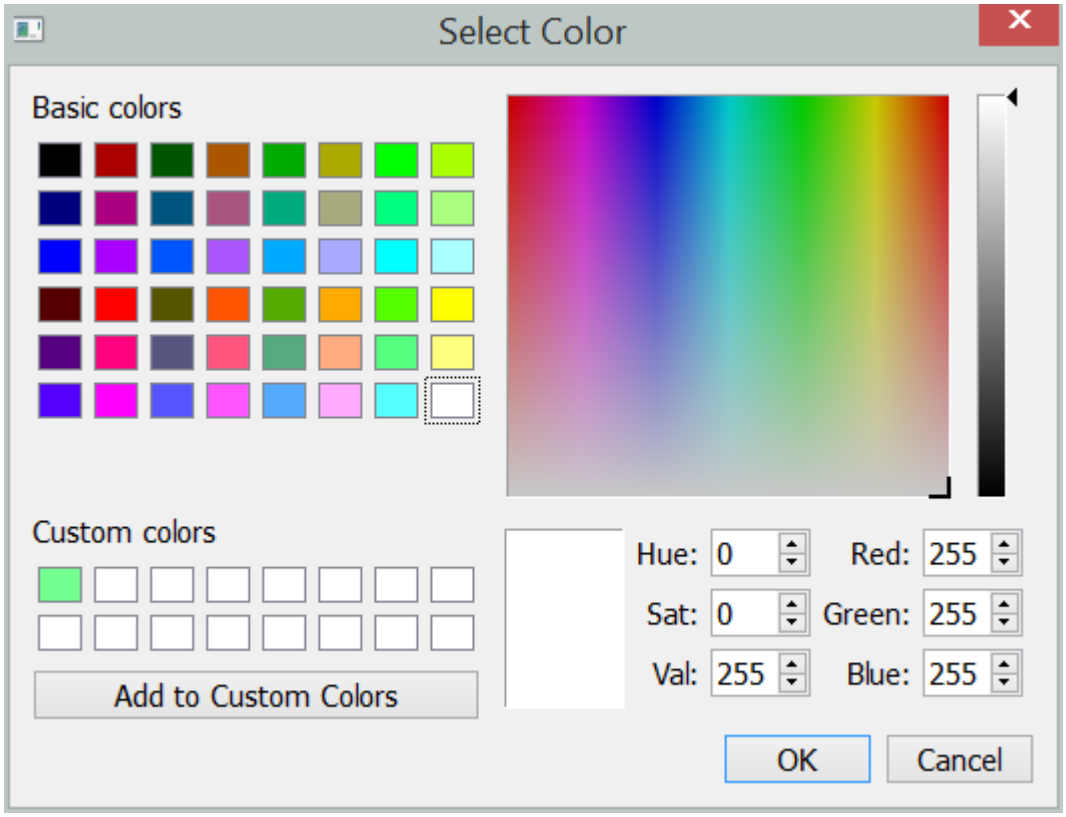
```
>>> import easygui_qt as easy
>>> choice = easy.get_continue_or_cancel()
```



easygui_qt.easygui_qt.**get_color_hex**()

Using a color dialog, returns a color in hexadecimal notation i.e. a string '#RRGGBB' or "None" if color dialog is dismissed.

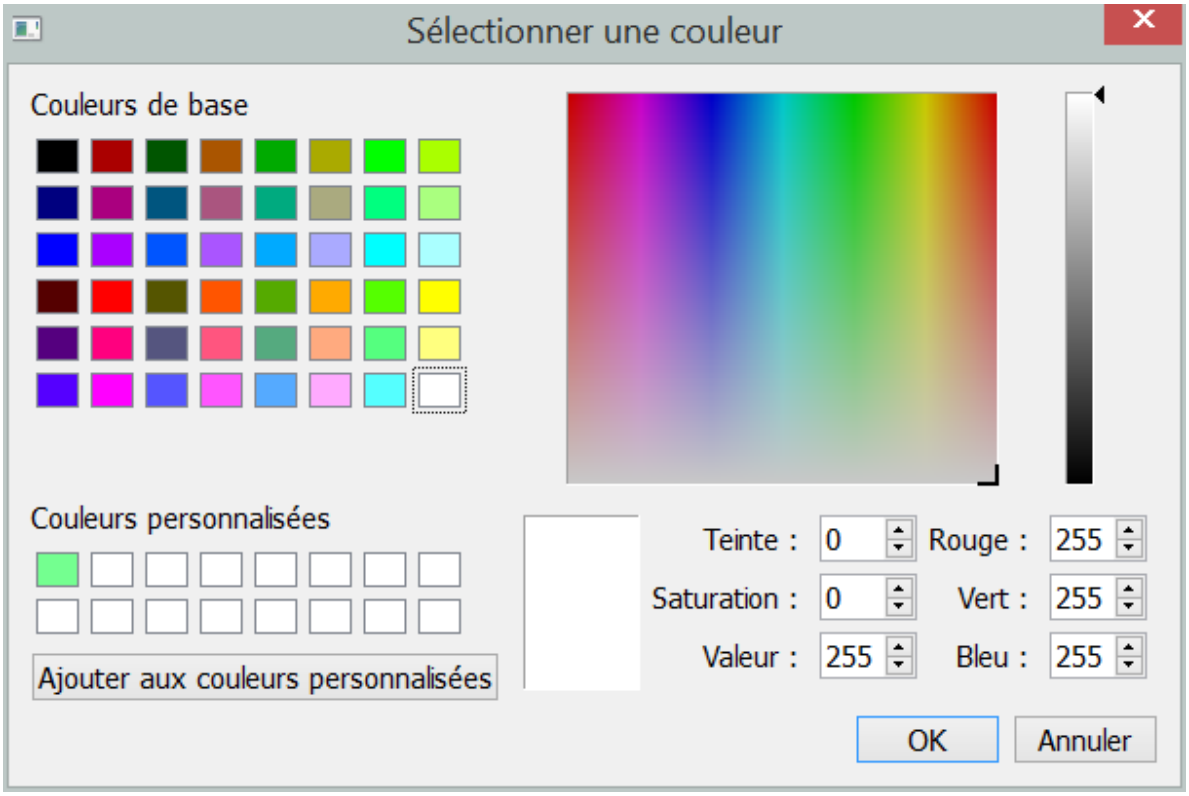
```
>>> import easygui_qt as easy
>>> color = easy.get_color_hex()
```



easygui_qt.easygui_qt.**get_color_rgb**(*app=None*)

Using a color dialog, returns a color in rgb notation i.e. a tuple (r, g, b) or "None" if color dialog is dismissed.

```
>>> import easygui_qt as easy
>>> easy.set_language('fr')
>>> color = easy.get_color_rgb()
```



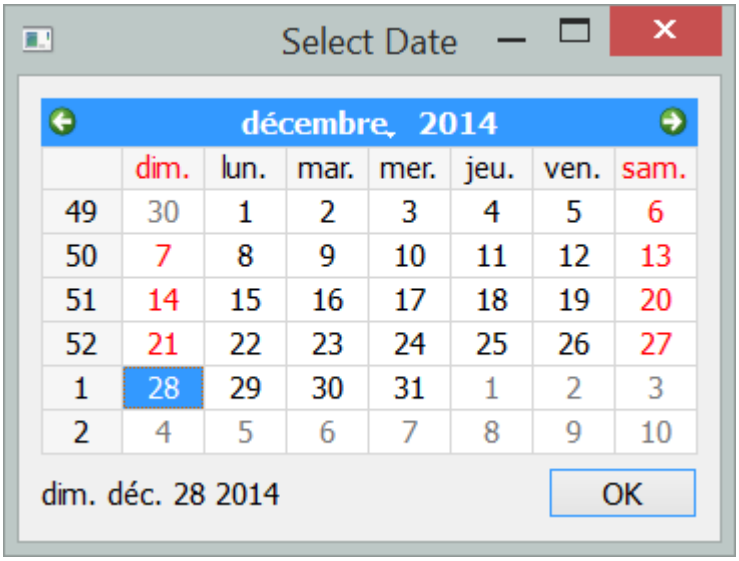
`easygui_qt.easygui_qt.`**`get_date`**(*title='Select Date'*)

Calendar widget

Parameters **`title`** – window title

Returns the selected date as a `datetime.date` instance

```
>>> import easygui_qt as easy
>>> date = easy.get_date()
```



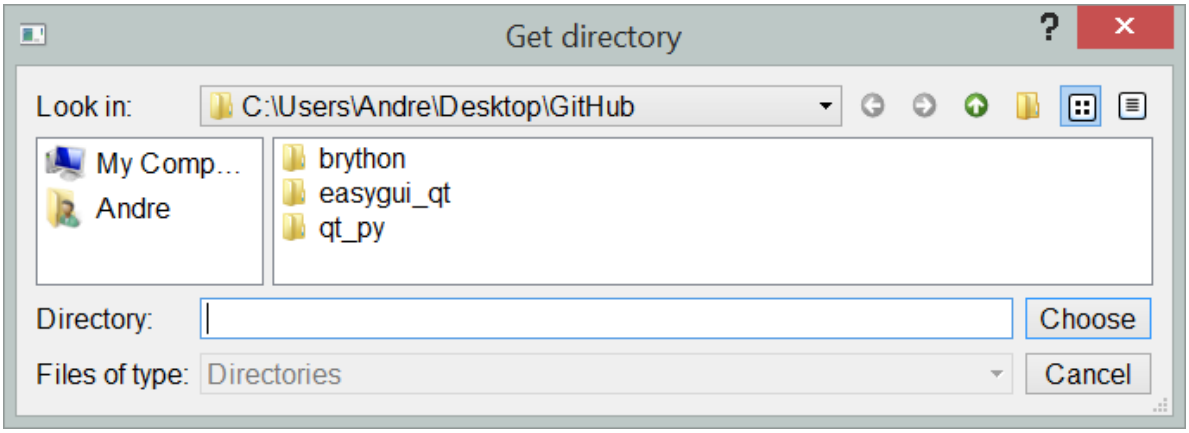
`easygui_qt.easygui_qt.`**`get_directory_name`**(*title='Get directory'*)

Gets the name (full path) of an existing directory

Parameters **title** – Window title

Returns the name of a directory or an empty string if cancelled.

```
>>> import easygui_qt as easy
>>> easy.get_directory_name()
```



By default, this dialog initially displays the content of the current working directory.

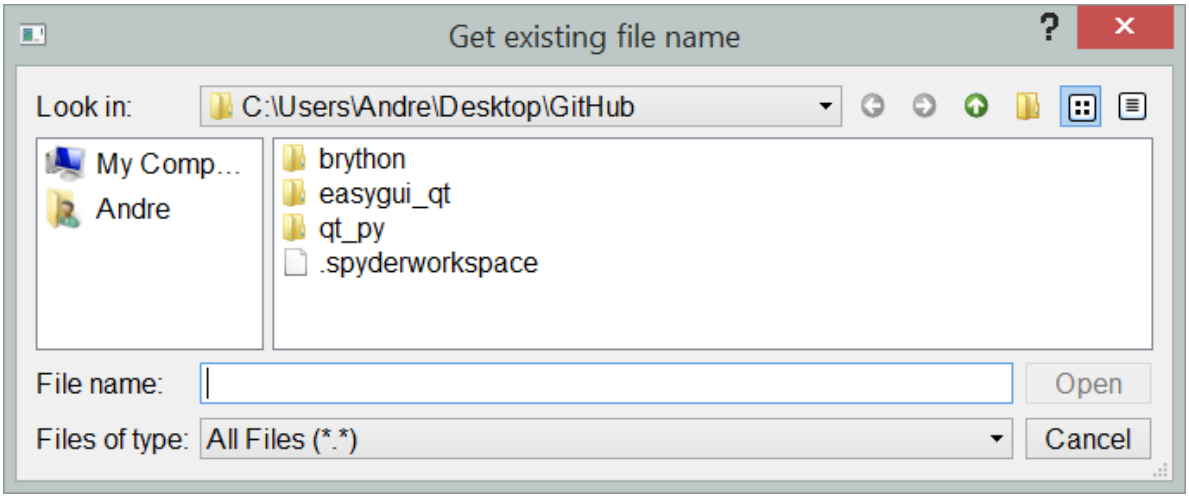
easygui_qt.easygui_qt.**get_file_names**(*title='Get existing file names'*)

Gets the names (full path) of existing files

Parameters **title** – Window title

Returns the list of names (paths) of files selected. (It can be an empty list.)

```
>>> import easygui_qt as easy
>>> easy.get_file_names()
```



By default, this dialog initially displays the content of the current working directory.

easygui_qt.easygui_qt.**get_save_file_name**(*title='File name to save'*)

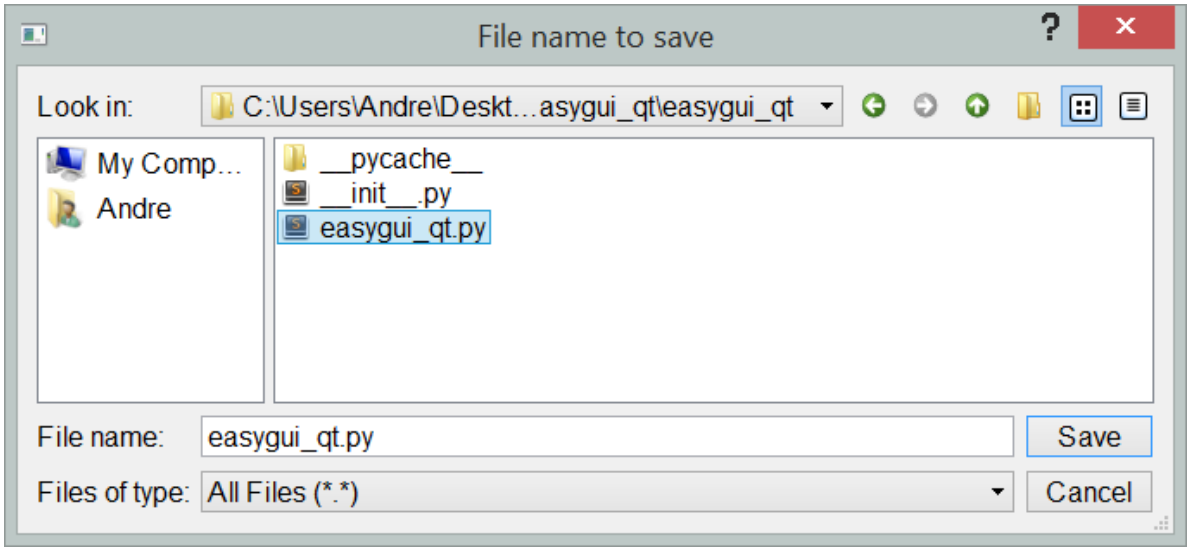
Gets the name (full path) of of a file to be saved.

Parameters **title** – Window title

Returns the name (path) of file selected

The user is warned if the file already exists and can choose to cancel. However, this dialog actually does NOT save any file: it only return a string containing the full path of the chosen file.

```
>>> import easygui_qt as easy
>>> easy.get_save_file_name()
```

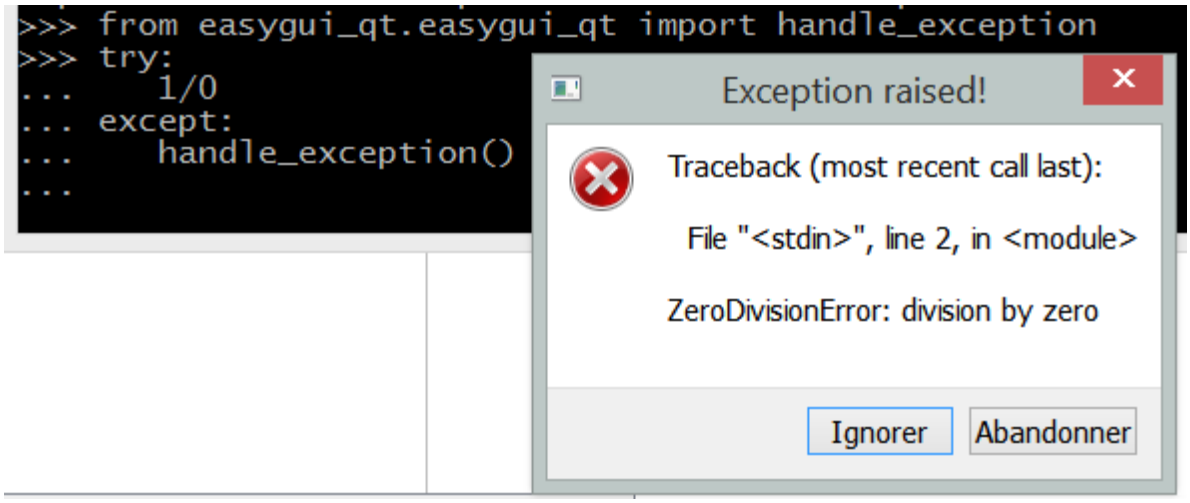


By default, this dialog initially displays the content of the current working directory.

easygui_qt.easygui_qt.**handle_exception**(*title='Exception raised!'*)

Displays a traceback in a window if an exception is raised. If the user clicks on "abort", sys.exit() is called and the program ends. If the user clicks on "ignore", the program resumes its execution.

Parameters **title** – the window title



easygui_qt.easygui_qt.**set_font_size**(*font_size*)

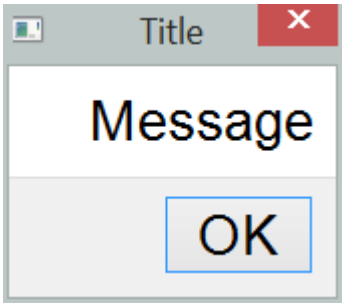
Simple method to set font size.

Parameters **font_size** – integer value

Does not create a GUI widget; but affects the appearance of future GUI widgets.

```
>>> import easygui_qt as easy
>>> easy.set_font_size(20)
```

```
>>> easy.show_message()
```



easygui_qt.easygui_qt.**get_language**(*title='Select language'*, *name='Language codes'*, *instruction=None*)

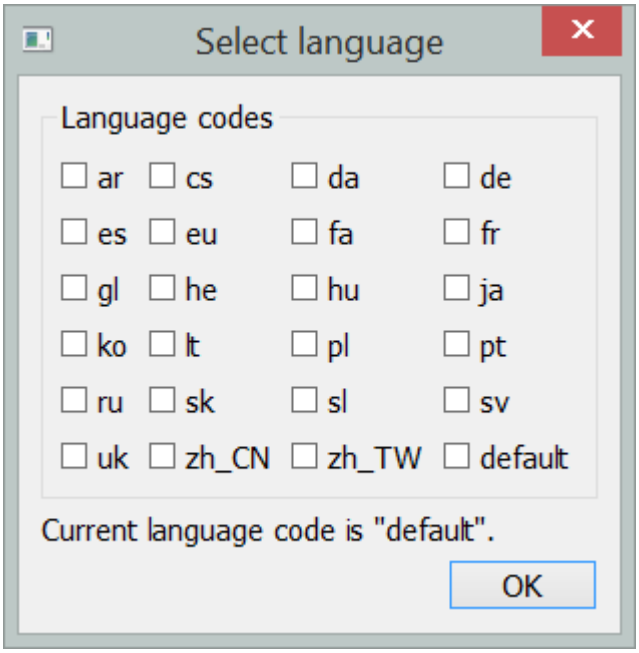
Dialog to choose language based on some locale code for files found on default path.

Parameters

- **title** – Window title
- **name** – Heading for valid values of locale appearing in checkboxes
- **instruction** – Like the name says; when set to None, a default string is used which includes the current language used.

The first time an EasyGUI_Qt widget is created in a program, the PyQt language files found in the standard location of the user's computer are scanned and recorded; these provide some translations of standard GUI components (like name of buttons). Note that "en" is not found as a locale (at least, not on the author's computer) but using "default" reverts the choice to the original (English here).

```
>>> import easygui_qt as easy
>>> easy.get_language()
```



easygui_qt.easygui_qt.**set_language**(*locale*)

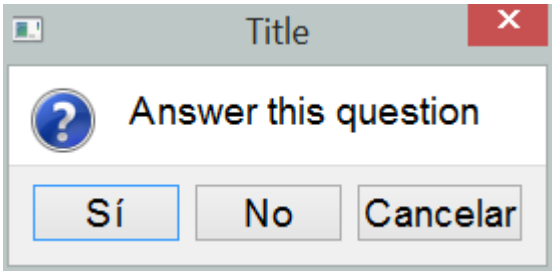
Sets the locale, if available

Parameters **locale** – standard code for locale (e.g. 'fr', 'en_CA')

Does not create a GUI widget, but affect the appearance of widgets created afterwards

```
>>> import easygui_qt as easy
>>> easy.set_locale('es')
```

```
>>> # after setting the locale
>>> easy.get_yes_or_no()
```



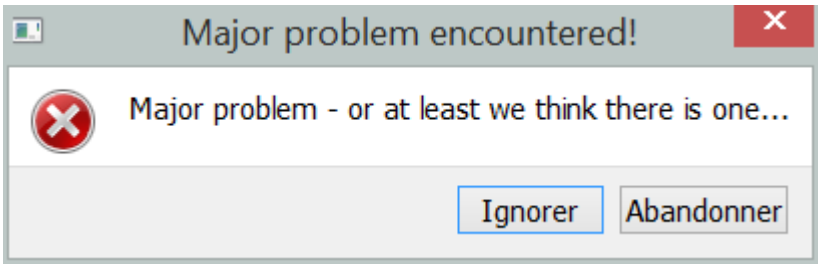
easygui_qt.easygui_qt.**get_abort**(*message='Major problem - or at least we think there is one...'*, *title='Major problem encountered!'*)

Displays a message about a problem. If the user clicks on "abort", sys.exit() is called and the program ends. If the user clicks on "ignore", the program resumes its execution.

Parameters

- **title** – the window title
- **message** – the message to display

```
>>> import easygui_qt as easy
>>> easy.get_abort()
```



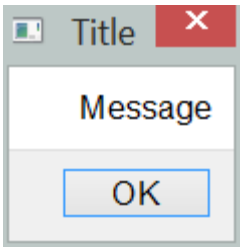
easygui_qt.easygui_qt.**show_message**(*message='Message'*, *title='Title'*)

Simple message box.

Parameters

- **message** – message string
- **title** – window title

```
>>> import easygui_qt as easy
>>> easy.show_message()
```



easygui_qt.easygui_qt.**show_file**(*file_name=None*, *title='Title'*, *file_type='text'*)

Displays a file in a window. While it looks as though the file can be edited, the only changes that happened are in the window and nothing can be saved.

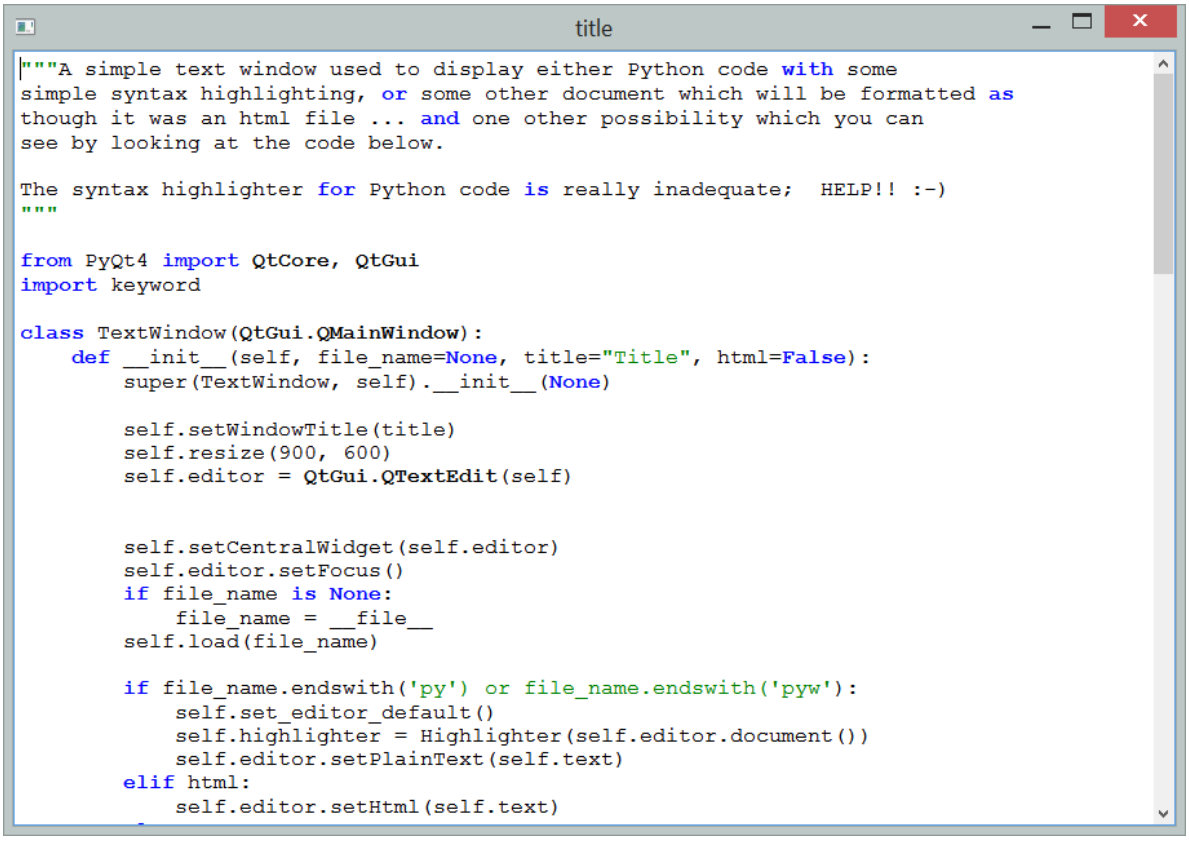
Parameters

- **title** – the window title
- **file_name** – the file name, (path) relative to the calling program
- **file_type** – possible values: `text`, `code`, `html`, `python`.

By default, file_type is assumed to be `text`; if set to `code`, the content is displayed with a monospace font and, if set to `python`, some code highlighting is done. If the file_type is `html`, it is processed assuming it follows html syntax.

Note: a better Python code hightlighter would be most welcome!

```
>>> import easygui_qt as easy
>>> easy.show_file()
```



easygui_qt.easygui_qt.**show_text**(*title='Title'*, *text=''*)

Displays some text in a window.

Parameters

- **title** – the window title
- **code** – a string to display in the window.

```
>>> import easygui_qt as easy
>>> easy.show_code()
```

../docs/images/show_text.png

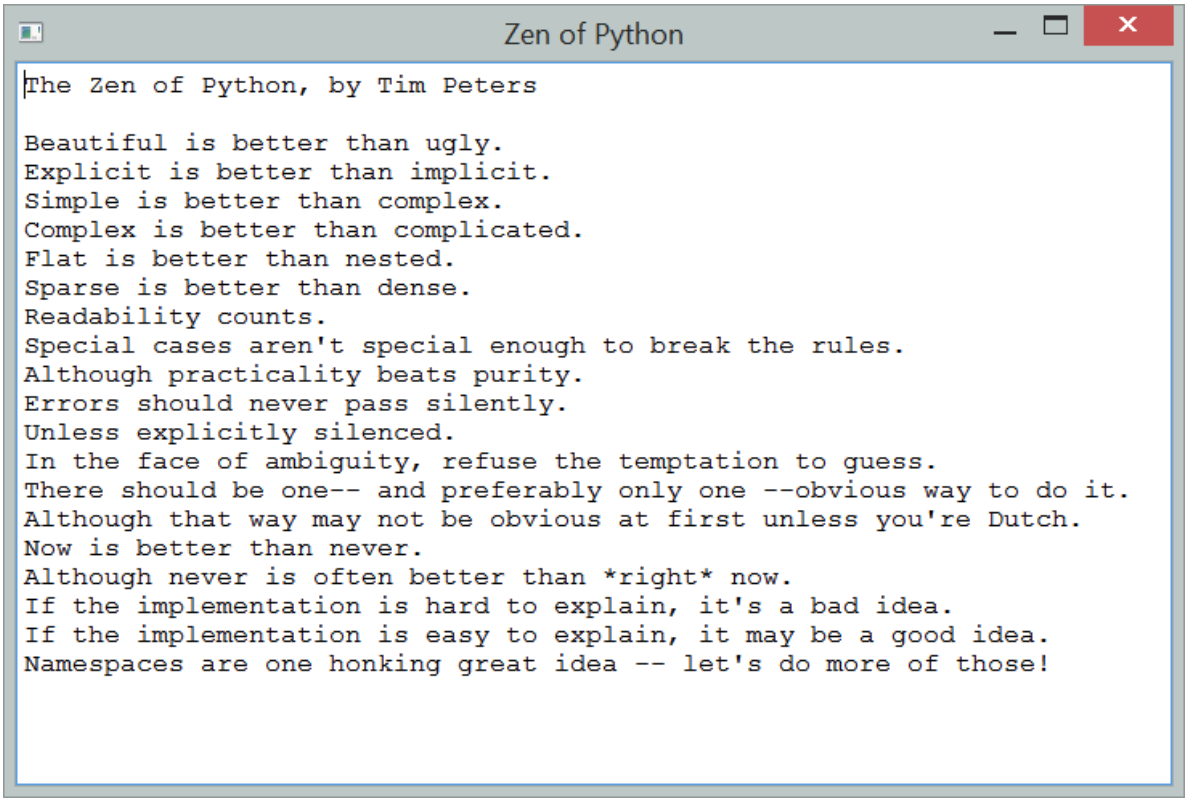
easygui_qt.easygui_qt.**show_code**(*title='Title'*, *text=''*)

Displays some text in a window, in a monospace font.

Parameters

- **title** – the window title
- **code** – a string to display in the window.

```
>>> import easygui_qt as easy
>>> easy.show_code()
```



easygui_qt.easygui_qt.**show_html**(*title='Title'*, *text=''*)

Displays some html text in a window.

Parameters

- **title** – the window title
- **code** – a string to display in the window.

```
>>> import easygui_qt as easy
>>> easy.show_html()
```

easygui_qt.easygui_qt.**find_help**()

Opens a web browser, pointing at the documention about EasyGUI_Qt available on the web.

CHAPTER 6

Comparison with easygui

EasyGUI_Qt was inspired by EasyGUI.

Here is a **brief** summary table of the corresponding function names for widgets with similar purpose being used in each project. This table is **not** complete, and is mainly provided to illustrate the different convention used when naming widgets. Please see the api for more details

EasyGUI	EasyGUI_Qt	Description
enterbox	get_string	Gets a string from the user
multenterbox	get_many_strings	User enters multiple values
integerbox	get_int *or* get_integer	Gets an integer from the user
	get_float	Gets a float from the user
msgbox	show_message	Displays a messages with an "ok" button
ccbox	get_continue_or_cancel	Choice: continue or cancel box
ynbox	get_yes_or_no	Answer a question with "yes" or "no" answer as choices
choicebox	get_choice	User selects a single choice from a list
multichoicebox	get_list_of_choices	User can select multiple choices from a list
passwordbox	get_password	Gets string from user, the text is masked as it is typed in
textbox	show_file	Displays text in proportional font, with word wrapping for EasyGUI / Displays text from a file, either in monospace font *or* formatted if html document
codebox	show_code	Displays text in monospace font, with no word wrapping
diropenbox	get_directory_name	Returns the name of a directory
fileopenbox	get_file_names	Returns the name of a file / list of files for EasyGUI_Qt
filesavebox	get_save_file_name	Returns the name of a file
exceptionbox	handle_exception	Displays a traceback

CHAPTER 7

Contributing

The following has been adapted from the boilerplate version created by cookiecutter. *Make sure you read the relevant parts as I do things a bit differently.* ;-)

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/aroberge/easygui_qt/issues.

If you are reporting a bug, please include:

- Have a look first at the existing issues (even the closed ones) - to avoid duplication.
- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.
- Screen captures can be useful.

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. You might want to have a look at https://github.com/aroberge/easygui_qt/issues/13

Something that would be **really nice** is to have unittest working that make use of QTest. I just have not been able to do this. I started implementing unittests using pyautogui but it (like Sikuli which could be another alternative) requires the windows to be left on their own while the tests are "slowly" executed; furthermore, it had been found to be unreliable on OSX with Python 2.7 (no report about OSX + Python 3+) as the windows appear "under" other already present.

Write Documentation

EasyGUI_Qt, like any project, could always use more documentation, whether as part of theofficial EasyGUI_Qt 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/aroberge/easygui_qt/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.
- Try to provide a specific use-case. Please note that some good ideas may not be implemented so as to keep the API easy to use for beginners.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

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

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

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

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

```
$ mkvirtualenv easygui_qt
$ cd easygui_qt/
$ 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. Make sure you check your code, running any tests or demos, and see that it follows PEP8. If you are adding a new widget, add it to the launcher demo.
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 ... well, if I get tests working properly. If it is a new widget, you should add it to the launcher and possibly creating a specific demo.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for at least Python 3.3, and 3.4. ... Ideally, it should also be tested with Python 3.2 and Python 2.7 as it would be nice to support these older version.

CHAPTER 8

Credits

Development Lead

- André Roberge <andre.roberge@gmail.com>

Contributors

- Jeremy R. Gray <jrgray@gmail.com>

CHAPTER 9

History

0.9.2

(Some of the changes noted are addition or improvements submitted by David Hughes via email)

- TextWindow now shows input either from a file or from a supplied string.
- added show_code()
- added show_text()
- renamed show() to show_message() [reverting change from 0.9.1]
- started creation of custom "page format" for more complex dialogs
- changed get_date() so that it returns a datetime date instance

Note: the documentation has NOT been updated to reflect these changes.

0.9.1

- removed verification from get_new_password
- added find_help
- created "back end" for wizard creator - will become show_story()
- documented and changed naming convention
- renamed select_language() : get_language()
- renamed show_message() : show()
- fixed a unicode bug for Python 2
- changed the way show_file works
- removed required_install PyQt4 from setup.py

0.9.0a

- Simplified the way change_password was implemented by reusing one of the new modules and fixed an unreported bug in the process
- changed the formatting of this file so that it should not cause problems with PyPI anymore.

0.9.0

Major change in version number as almost all the desired widgets for version 1.0 have been implemented.

Release notes:

Some unicode problems are likely present when using Python 2.7; the primary target is Python 3.3+ ... but we try to support earlier version as well.

Some problems are present with Mac OSX and Python 2.7 (only?)

- added show_abort
- added get_many_strings
- added handle_exception
- added show_code
- added show_file
- added get_new_password
- adressed an issue where some dialogs would appear below some windows (e.g. terminal) when launched from some platforms (e.g. Mac OSX): the goal should be that the dialogs always appear on top of other windows.
- removed with_app decorator; this decorator had been introduced to reduce the amount of repetitive code appearing in each function (and initially inspected the function signature to add automatically some additional keyword args) but it likely made it impossible to do unit testing with QTest (still not done) and prevented ReadTheDocs from reading the correct signatures for the decorated functions.
- tooltips added to demos launcher
- added get_username_password

0.4.0

- added get_password
- added get_date
- added get_color_hex
- added get_color_rgb
- added get_continue_or_cancel
- added roadmap as a github issue https://github.com/aroberge/easygui_qt/issues/13
- removed CONFIG as a global dict; using the configuration file instead.

- remove set_default_font
- rename set_locale to set_language
- added configuration file to save locale and font size

0.3.0

- Decided to support (with lower priority) Python 2 (2.7.9 more specifically)
- Should work reasonably well with Python 2.7.9 - other than potential unicode related issues
- made get_list_of_choices(), get_choice(), get_string(), and get_directory_name() work properly with Python 2.7.9

0.2.3a

- changed extension of some demos (from .pyw to .py) as they were not uploaded to pypi

0.2.3

- added demos dir to setup.py so that it can be included on pypi

0.2.2a

- changing path on image in readme in attempt to help pypi display properly

0.2.2

- changed the syntax for calls to super() to be compatible with Python 2. Note that the intention is to be a Python 3 project, but if simple changes can make it compatible with Python 2, they will be incorporated.
- changed name of set_save_file_name to get_save_file_name
- changed name of yes_no_question to get_yes_or_no
- added get_list_of_choices
- added demo launcher

0.2.1

- Moved the demos directory to a more sensible location
- added get_directory_name
- added get_file_names
- added set_save_file_name

- attempt to fix bug for Python 3.2 where inspect.signature was not defined

0.2.0

The API has been changed since the initial release and the following widgets have been documented, with images inserted in the documentation.

- get_choice
- get_float
- get_int
- get_integer
- get_string
- set_font_size
- set_default_font
- select_language
- set_locale
- show_message
- yes_no_question

0.1.0

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index