easygui Documentation

Release 0.98.0-RELEASED

easygui dev team

October 29, 2016

EasyGUI is a module for very simple, very easy GUI programming in Python. EasyGUI is different from other GUI generators in that EasyGUI is NOT event-driven. Instead, all GUI interactions are invoked by simple function calls.

EasyGui provides an easy-to-use interface for simple GUI interaction with a user. It does not require the programmer to know anything about tkinter, frames, widgets, callbacks or lambda.

EasyGUI runs on Python 2 and 3, and does not have any dependencies.

Example Usage

```
>>> import easygui
>>> easygui.ynbox('Shall I continue?', 'Title', ('Yes', 'No'))
True
>>> easygui.msgbox('This is a basic message box.', 'Title Goes Here')
'OK'
>>> easygui.buttonbox('Click on your favorite flavor.', 'Favorite Flavor', ('Chocolate', 'Vanilla',
'Chocolate'
```

How to get easygui

The best method to get easygui on your system is to type:

```
pip install --upgrade easygui
```

, which will install the latest easygui. You may also download the file yourself by looking for the latest release in sourceforge:

sourceforge download

CHAPTER 3

Table of Contents

3.1 EasyGui Support and Contacts

3.1.1 Getting easygui and getting help

The easygui project is developed on Github: https://github.com/robertlugg/easygui

Releases are typically put in both sourceforge and pypi: http://sourceforge.net/projects/easygui

You can use either place to download the latest version, submit bugs, and make requests for improvements.

We welcome any and all feedback!

3.1.2 Help develop easygui

If you want to delve into the inner workings of easygui, wish to preview the next release, or if you want to contribute to develop easygui, feel free to explore our GitHub site

You very welcome to fork and please let us know when you come up with cool stuff others might use!

3.1.3 Thanks

The following people (as well as many others) have contributed to easygui. Thank you for your work!

Alex Zawadzki - Project Management, Design, Testing

Horst JENS - Design, Documentation, Testing

Juan Jose Corrales - Development (Active)

Robert Lugg - Development (Active)

Stephen Ferg - (retired) Created and developed easygui through 0.96

Andreas Noteng - Debian Package Manager

3.2 easygui API

easygui.**buttonbox**(*msg=''*, *title=' '*, *choices=('Button[1]', 'Button[2]', 'Button[3]')*, *image=None*, *images=None*, *default_choice=None*, *cancel_choice=None*, *callback=None*)

Display a msg, a title, an image, and a set of buttons. The buttons are defined by the members of the choices global_state.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **tuple, dict choices** (*list,*) – a list or tuple of the choices to be displayed
- **image** (*str*) – (Only here for backward compatibility)
- **list images** (*str,*) – Filename of image or iterable or iterable of iterable to display
- **default_choice** (*str*) – The choice you want highlighted when the gui appears
- **cancel_choice** (*str*) – This choice will close the window, justa as Escape key or x
- **callback** (*function*) – A callback function to be called when a choice button is pressed

Returns the text of the button that the user selected, or the value if it was a dictionary

easygui.**diropenbox**(*msg=None*, *title=None*, *default=None*)

A dialog to get a directory name. Note that the msg argument, if specified, is ignored.

Returns the name of a directory, or None if user chose to cancel.

If the "default" argument specifies a directory name, and that directory exists, then the dialog box will start with that directory.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **default** (*str*) – starting directory when dialog opens

Returns Normalized path selected by user

easygui.**fileopenbox**(*msg=None*, *title=None*, *default='*'*, *filetypes=None*, *multiple=False*)

A dialog to get a file name.

About the "default" argument

The "default" argument specifies a filepath that (normally) contains one or more wildcards. fileopenbox will display only files that match the default filepath. If omitted, defaults to "*" (all files in the current directory).

WINDOWS EXAMPLE:

```
...default="c:/myjunk/*.py"
```

will open in directory c:myjunkand show all Python files.

WINDOWS EXAMPLE:

```
...default="c:/myjunk/test*.py"
```

will open in directory c:myjunkand show all Python files whose names begin with "test".

Note that on Windows, fileopenbox automatically changes the path separator to the Windows path separator (backslash).

About the "filetypes" argument

If specified, it should contain a list of items, where each item is either:

- a string containing a filemask # e.g. "*.txt"
- a list of strings, where all of the strings except the last one are filemasks (each beginning with "*.", such as "*.txt" for text files, "*.py" for Python files, etc.). and the last string contains a filetype description

EXAMPLE:

```
filetypes = ["*.css", ["*.htm", "*.html", "HTML files"]  ]
```

Note: If the filetypes list does not contain ("All files","*"), it will be added.

If the filetypes list does not contain a filemask that includes the extension of the "default" argument, it will be added. For example, if default="*abc.py" and no filetypes argument was specified, then "*.py" will automatically be added to the filetypes argument.

Parameters

- **msg** (*str*) – the msg to be displayed.
- **title** (*str*) – the window title
- **default** (*str*) – filepath with wildcards
- **filetypes** (*object*) – filemasks that a user can choose, e.g. "*.txt"
- **multiple** (*bool*) – If true, more than one file can be selected

Returns the name of a file, or None if user chose to cancel

`easygui.`**filesavebox**(*msg=None*, *title=None*, *default=''*, *filetypes=None*)

A file to get the name of a file to save. Returns the name of a file, or None if user chose to cancel.

The "default" argument should contain a filename (i.e. the current name of the file to be saved). It may also be empty, or contain a filemask that includes wildcards.

The "filetypes" argument works like the "filetypes" argument to fileopenbox.

Parameters

- **msg** (*str*) – the msg to be displayed.
- **title** (*str*) – the window title
- **default** (*str*) – default filename to return
- **filetypes** (*object*) – filemasks that a user can choose, e.g. " *.txt"

Returns the name of a file, or None if user chose to cancel

`easygui.`**textbox**(*msg=''*, *title=' '*, *text=''*, *codebox=False*, *callback=None*, *run=True*)

Display a message and a text to edit

msg [string] text displayed in the message area (instructions...)

title [str] the window title

text: str, list or tuple text displayed in textAreas (editable)

codebox: bool if True, don't wrap and width is set to 80 chars

callback: function if set, this function will be called when OK is pressed

run: bool if True, a box object will be created and returned, but not run

None If cancel is pressed

str If OK is pressed returns the contents of textArea

`easygui.`**`ynbox`**(*msg='Shall I continue?', title=' ', choices=('[<F1>]Yes', '[<F2>]No'), image=None, default_choice='[<F1>]Yes', cancel_choice='[<F2>]No'*)

Display a msgbox with choices of Yes and No.

The returned value is calculated this way:

```
if the first choice ("Yes") is chosen, or if the dialog is cancelled:
    return True
else:
    return False
```

If invoked without a msg argument, displays a generic request for a confirmation that the user wishes to continue. So it can be used this way:

```
if ynbox():
    pass # continue
else:
    sys.exit(0)   # exit the program
```

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **choices** (*list*) – a list or tuple of the choices to be displayed
- **image** (*str*) – Filename of image to display
- **default_choice** (*str*) – The choice you want highlighted when the gui appears
- **cancel_choice** (*str*) – If the user presses the 'X' close, which button should be pressed

Returns True if 'Yes' or dialog is cancelled, False if 'No'

`easygui.`**`ccbox`**(*msg='Shall I continue?', title=' ', choices=('C[o]ntinue', 'C[a]ncel'), image=None, default_choice='Continue', cancel_choice='Cancel'*)

Display a msgbox with choices of Continue and Cancel.

The returned value is calculated this way:

```
if the first choice ("Continue") is chosen,
  or if the dialog is cancelled:
    return True
else:
    return False
```

If invoked without a msg argument, displays a generic request for a confirmation that the user wishes to continue. So it can be used this way:

```
if ccbox():
    pass # continue
else:
    sys.exit(0)   # exit the program
```

Parameters

- **msg** (*str*) – the msg to be displayed

- **title** (*str*) – the window title
- **choices** (*list*) – a list or tuple of the choices to be displayed
- **image** (*str*) – Filename of image to display
- **default_choice** (*str*) – The choice you want highlighted when the gui appears
- **cancel_choice** (*str*) – If the user presses the 'X' close, which button should be pressed

Returns True if 'Continue' or dialog is cancelled, False if 'Cancel'

easygui.**boolbox**(*msg='Shall I continue?', title=' ', choices=('[Y]es', '[N]o'), image=None, default_choice='Yes', cancel_choice='No'*)

Display a boolean msgbox.

The returned value is calculated this way:

```
if the first choice is chosen, or if the dialog is cancelled:
    returns True
else:
    returns False
```

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **choices** (*list*) – a list or tuple of the choices to be displayed
- **image** (*str*) – Filename of image to display
- **default_choice** (*str*) – The choice you want highlighted when the gui appears
- **cancel_choice** (*str*) – If the user presses the 'X' close, which button should be pressed

Returns True if first button pressed or dialog is cancelled, False if second button is pressed

easygui.**indexbox**(*msg='Shall I continue?', title=' ', choices=('Yes', 'No'), image=None, default_choice='Yes', cancel_choice='No'*)

Display a buttonbox with the specified choices.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **choices** (*list*) – a list or tuple of the choices to be displayed
- **image** (*str*) – Filename of image to display
- **default_choice** (*str*) – The choice you want highlighted when the gui appears
- **cancel_choice** (*str*) – If the user presses the 'X' close, which button should be pressed

Returns the index of the choice selected, starting from 0

easygui.**msgbox**(*msg='(Your message goes here)', title=' ', ok_button='OK', image=None, root=None*)

Display a message box

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title

- **ok_button** (*str*) – text to show in the button
- **image** (*str*) – Filename of image to display
- **root** (*tk_widget*) – Top-level Tk widget

Returns the text of the ok_button

easygui.**integerbox**(*msg=''*, *title=' '*, *default=None*, *lowerbound=0*, *upperbound=99*, *image=None*, *root=None*)

Show a box in which a user can enter an integer.

In addition to arguments for msg and title, this function accepts integer arguments for "default", "lowerbound", and "upperbound".

The default, lowerbound, or upperbound may be None.

When the user enters some text, the text is checked to verify that it can be converted to an integer between the lowerbound and upperbound.

If it can be, the integer (not the text) is returned.

If it cannot, then an error msg is displayed, and the integerbox is redisplayed.

If the user cancels the operation, None is returned.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **default** (*int*) – The default value to return
- **lowerbound** (*int*) – The lower-most value allowed
- **upperbound** (*int*) – The upper-most value allowed
- **image** (*str*) – Filename of image to display
- **root** (*tk_widget*) – Top-level Tk widget

Returns the integer value entered by the user

easygui.**multenterbox**(*msg='Fill in values for the fields.'*, *title=' '*, *fields=[]*, *values=[]*, *callback=None*, *run=True*)

Show screen with multiple data entry fields.

If there are fewer values than names, the list of values is padded with empty strings until the number of values is the same as the number of names.

If there are more values than names, the list of values is truncated so that there are as many values as names.

Returns a list of the values of the fields, or None if the user cancels the operation.

Here is some example code, that shows how values returned from multenterbox can be checked for validity before they are accepted:

```
msg = "Enter your personal information"
title = "Credit Card Application"
fieldNames = ["Name","Street Address","City","State","ZipCode"]
fieldValues = []  # we start with blanks for the values
fieldValues = multenterbox(msg,title, fieldNames)

# make sure that none of the fields was left blank
while 1:
    if fieldValues is None: break
```

```
        errmsg = ""
        for i in range(len(fieldNames)):
            if fieldValues[i].strip() == "":
                errmsg += ('"%s" is a required field.\n\n' % fieldNames[i])
        if errmsg == "":
            break # no problems found
        fieldValues = multenterbox(errmsg, title, fieldNames, fieldValues)

    print("Reply was: %s" % str(fieldValues))
```

Parameters

- **msg** (*str*) – the msg to be displayed.
- **title** (*str*) – the window title
- **fields** (*list*) – a list of fieldnames.
- **values** (*list*) – a list of field values

Returns String

easygui.**enterbox**(*msg='Enter something.', title=' ', default=' ', strip=True, image=None, root=None*)
Show a box in which a user can enter some text.

You may optionally specify some default text, which will appear in the enterbox when it is displayed.

Example:

```
    reply = enterbox(....)
    if reply:
        ...
    else:
        ...
```

Parameters

- **msg** (*str*) – the msg to be displayed.
- **title** (*str*) – the window title
- **default** (*str*) – value returned if user does not change it
- **strip** (*bool*) – If True, the return value will have its whitespace stripped before being returned

Returns the text that the user entered, or None if he cancels the operation.

easygui.**exceptionbox**(*msg=None, title=None*)
Display a box that gives information about an exception that has just been raised.

The caller may optionally pass in a title for the window, or a msg to accompany the error information.

Note that you do not need to (and cannot) pass an exception object as an argument. The latest exception will automatically be used.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title

Returns None

easygui.**choicebox**(*msg='Pick an item'*, *title=''*, *choices=[]*, *preselect=0*, *callback=None*, *run=True*)

Present the user with a list of choices. return the choice that he selects.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **choices** (*list*) – a list or tuple of the choices to be displayed
- **preselect** – Which item, if any are preselected when dialog appears

Returns List containing choice selected or None if cancelled

easygui.**codebox**(*msg=''*, *title=' '*, *text=''*)

Display some text in a monospaced font, with no line wrapping. This function is suitable for displaying code and text that is formatted using spaces.

The text parameter should be a string, or a list or tuple of lines to be displayed in the textbox.

Parameters

- **msg** (*str*) – the msg to be displayed
- **title** (*str*) – the window title
- **text** (*str*) – what to display in the textbox

easygui.**passwordbox**(*msg='Enter your password.'*, *title=' '*, *default=''*, *image=None*, *root=None*)

Show a box in which a user can enter a password. The text is masked with asterisks, so the password is not displayed.

Parameters

- **msg** (*str*) – the msg to be displayed.
- **title** (*str*) – the window title
- **default** (*str*) – value returned if user does not change it

Returns the text that the user entered, or None if he cancels the operation.

easygui.**multpasswordbox**(*msg='Fill in values for the fields.'*, *title=' '*, *fields=()*, *values=()*, *callback=None*, *run=True*)

Same interface as multenterbox. But in multpassword box, the last of the fields is assumed to be a password, and is masked with asterisks.

Parameters

- **msg** (*str*) – the msg to be displayed.
- **title** (*str*) – the window title
- **fields** (*list*) – a list of fieldnames.
- **values** (*list*) – a list of field values

Returns String

Example

Here is some example code, that shows how values returned from multpasswordbox can be checked for validity before they are accepted:

```
msg = "Enter logon information"
title = "Demo of multpasswordbox"
fieldNames = ["Server ID", "User ID", "Password"]
fieldValues = []  # we start with blanks for the values
fieldValues = multpasswordbox(msg,title, fieldNames)

# make sure that none of the fields was left blank
while 1:
    if fieldValues is None: break
    errmsg = ""
    for i in range(len(fieldNames)):
        if fieldValues[i].strip() == "":
            errmsg = errmsg + ('"%s" is a required field.\n\n' %
             fieldNames[i])
        if errmsg == "": break # no problems found
    fieldValues = multpasswordbox(errmsg, title,
      fieldNames, fieldValues)

print("Reply was: %s" % str(fieldValues))
```

easygui.**multchoicebox**(*msg='Pick an item', title='', choices=[], preselect=0, callback=None, run=True*)

Same as choicebox, but the user can select many items.

class easygui.**EgStore**(*filename*)

Bases: `object`

A class to support persistent storage.

You can use `EgStore` to support the storage and retrieval of user settings for an EasyGui application.

First: define a class named Settings as a subclass of EgStore

```
class Settings(EgStore):
    def __init__(self, filename):  # filename is required
        # specify default values for variables that this application wants to remember
        self.user_id = ''
        self.target_server = ''
        settings.restore()
```

*Second: create a persistent Settings object**

```
settings = Settings('app_settings.txt')
settings.user_id = 'obama_barak'
settings.targetServer = 'whitehouse1'
settings.store()

# run code that gets a new value for user_id, and persist the settings
settings.user_id = 'biden_joe'
settings.store()
```

Example C: recover the Settings instance, change an attribute, and store it again.

```
settings = Settings('app_settings.txt')
settings.restore()
print settings
settings.user_id = 'vanrossum_g'
settings.store()
```

kill()

Delete this store's file if it exists.

restore()

store()

Save this store to a pickle file. All directories in `filename` must already exist.

easygui.**abouteasygui**()

shows the easygui revision history

easygui.**egdemo**()

Run the EasyGui demo.

3.3 EasyGui Tutorial

3.3.1 Introduction

In easygui, all GUI interactions are invoked by simple function calls.

Here is a simple demo program using easygui.

```
from easygui import *
import sys

# A nice welcome message
ret_val = msgbox("Hello, World!")
if ret_val is None: # User closed msgbox
    sys.exit(0)

msg ="What is your favorite flavor?\nOr Press <cancel> to exit."
title = "Ice Cream Survey"
choices = ["Vanilla", "Chocolate", "Strawberry", "Rocky Road"]
while 1:
    choice = choicebox(msg, title, choices)
    if choice is None:
        sys.exit(0)
    msgbox("You chose: {}".format(choice), "Survey Result")
```

3.3.2 EasyGui's demonstration routine

To run EasyGui's demonstration routine, invoke EasyGui from the command line this way:

```
python easygui.py
```

or from an IDE (such as IDLE, PythonWin, Wing, etc.) this way:

```
from easygui import *
egdemo()
```

This will allow you to try out the various EasyGui functions, and will print the results of your choices to the console.

3.3.3 Importing EasyGui

In order to use EasyGui, you must import it. The simplest import statement is:

```
import easygui
```

If you use this form, then to access the EasyGui functions, you must prefix them with the name "easygui", this way:

```
easygui.msgbox(...)
```

One alternative is to import EasyGui this way:

```
from easygui import *
```

This makes it easier to invoke the EasyGui functions; you won't have to prefix the function names with "easygui". You can just code something like this:

```
msgbox(...)
```

A third alternative is to use something like the following import statement:

```
import easygui as g
```

This allows you to keep the EasyGui namespace separate with a minimal amount of typing. You can access easgui functions like this:

```
g.msgbox(...)
```

This third alterative is actually the best way to do it once you get used to python and easygui.

3.3.4 Using EasyGui

Once your module has imported EasyGui, GUI operations are a simple a matter of invoking EasyGui functions with a few parameters. For example, using EasyGui, the famous "Hello, world!" program looks like this:

```
from easygui import *
msgbox("Hello, world!")
```

To see a demo of what EasyGui output looks like, invoke easyGui from the command line, this way:

```
python easygui.py
```

To see examples of code that invokes the EasyGui functions, look at the demonstration code at the end of easygui.py.

3.3.5 Default arguments for EasyGui functions

For all of the boxes, the first two arguments are for message and title, in that order. In some cases, this might not be the most user-friendly arrangement (for example, the dialogs for getting directory and filenames ignore the message argument), but I felt that keeping this consistent across all widgets was a consideration that is more important.

Most arguments to EasyGui functions have defaults. Almost all of the boxes display a message and a title. The title defaults to the empty string, and the message usually has a simple default.

This makes it is possible to specify as few arguments as you need in order to get the result that you want. For instance, the title argument to msgbox is optional, so you can call msgbox specifying only a message, this way:

```
msgbox("Danger, Will Robinson!")
```

or specifying a message and a title, this way:

```
msgbox("Danger, Will Robinson!", "Warning!")
```

On the various types of buttonbox, the default message is "Shall I continue?", so you can (if you wish) invoke them without arguments at all. Here we invoke ccbox (the close/cancel box, which returns a boolean value) without any arguments at all:

```
if ccbox():
    pass  # user chose to continue
else:
    return      # user chose to cancel
```

3.3.6 Using keyword arguments when calling EasyGui functions

It is possible to use keyword arguments when calling EasyGui functions.

Suppose for instance that you wanted to use a buttonbox, but (for whatever reason) did not want to specify the title (second) positional argument. You could still specify the choices argument (the third argument) using a keyword, this way:

```
choices = ["Yes","No","Only on Friday"]
reply = choicebox("Do you like to eat fish?", choices=choices)
```

3.3.7 Using buttonboxes

There are a number of functions built on top of buttonbox() for common needs.

msgbox

msgbox displays a message and offers an OK button. You can send whatever message you want, along with whatever title you want. You can even over-ride the default text of "OK" on the button if you wish. Here is the signature of the msgbox function:

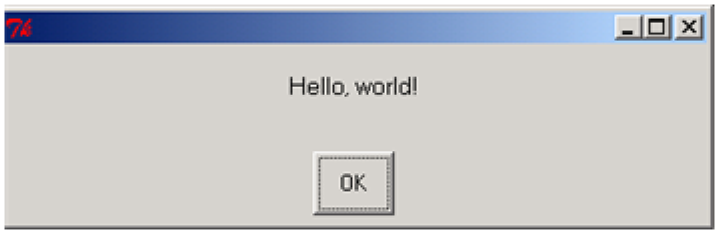
```
def msgbox(msg="(Your message goes here)", title="", ok_button="OK"):
    ....
```

The clearest way to over-ride the button text is to do it with a keyword argument, like this:

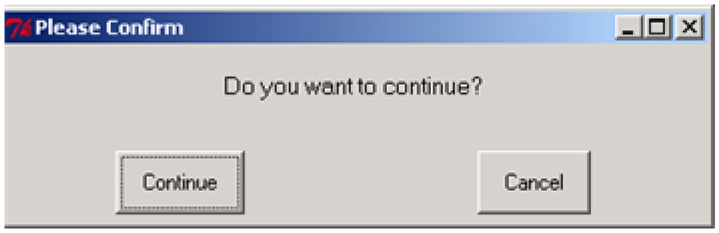
```
msgbox("Backup complete!", ok_button="Good job!")
```

Here are a couple of examples:

```
msgbox("Hello, world!")
```



```
msg = "Do you want to continue?"
title = "Please Confirm"
if ccbox(msg, title):     # show a Continue/Cancel dialog
    pass  # user chose Continue
else:  # user chose Cancel
    sys.exit(0)
```



Do you want to continue?

ccbox

ccbox offers a choice of Continue and Cancel, and returns either True (for continue) or False (for cancel).

ynbox

ynbox offers a choice of Yes and No, and returns either True of False.

buttonbox

To specify your own set of buttons in a buttonbox, use the buttonbox() function.

The buttonbox can be used to display a set of buttons of your choice. When the user clicks on a button, buttonbox() returns the text of the choice. If the user cancels or closes the buttonbox, the default choice (the first choice) is returned.

buttonbox displays a message, a title, and a set of buttons. Returns the text of the button that the user selected.

Choices can be of different types:

```
choices=["Alice", "Bob", "Charlie"]
buttonbox(msg="Choose a name", choices=choices)
# This would return a string: "Alice", "Bob" or "Charlie"

choices=("Alice", "Bob", "Charlie")
buttonbox(msg="Choose a name", choices=choices)
# This would return a string: "Alice", "Bob" or "Charlie"

choices = [("Alice", 1), ("Bob", 2), ("Charlie", 3)]
buttonbox(msg="Choose a name", choices=choices)
# This would return an integer (1, 2 or 3)

choices = [("Alice", 1), ("Bob", 2), ("Charlie", 3)]
choices = collections.OrderedDict(choices)
buttonbox(msg="Choose a name", choices=choices)
# This would return an integer (1, 2 or 3)

choices = {"Alice": 1), "Bob": 2, "Charlie", 3}
buttonbox(msg="Choose a name", choices=choices)
# This would return an integer (1, 2 or 3)
# But buttons would show an incorrect order
```

Callbacks with buttonbox

If the same box is going to be used again and again, it is nice to have a callback, let us see an example:

```
def update(box):
    number = random.randint(1, 10)
    msg = "This demoes interfacing WITH a callback. \nYou got number {} \nNotice the absence of flick
    box.set_msg(msg)

initial_msg = "This demoes interfacing with a callback \nPush to get a random number between 1 and 1(

buttonbox(
    title="This demoes interfacing with a callback",
    msg=initual_msg,
    choices=["Get me a number", "[C]ancel"],
    default_choice="Get me a number",
    cancel_choice="[C]ancel",
    callback=update)
```

The callback will be called everytime the user pushes any of the option buttons, and it receives a box object with four methods:

- set_msg(msg), changes the message
- stop(), stops the ui
- get_selected_choice(), informs of the button pushed
- get_selected_row_column(), informs of the image pushed

indexbox

indexbox displays a message, a title, and a set of buttons. Returns the index of the user's choice. For example, if you invoked index box with three choices (A, B, C), indexbox would return 0 if the user picked A, 1 if he picked B, and 2 if he picked C.

boolbox

boolbox (boolean box) displays a message, a title, and a set of buttons. Returns returns 1 if the first button is chosen. Otherwise returns 0.

Here is a simple example of a boolbox():

```
message = "What does she say?"
title = ""
if boolbox(message, title, ["She loves me", "She loves me not"]):
    sendher("Flowers") # This is just a sample function that you might write.
else:
    pass
```

3.3.8 How to show an image in a buttonbox

When you invoke the buttonbox function (or other functions that display a button box, such as msgbox, indexbox, ynbox, etc.), you can specify the keyword argument image=xxx where xxx is the filename of an image. The file can be .gif.

Usually, you can use other image formats such as .png.

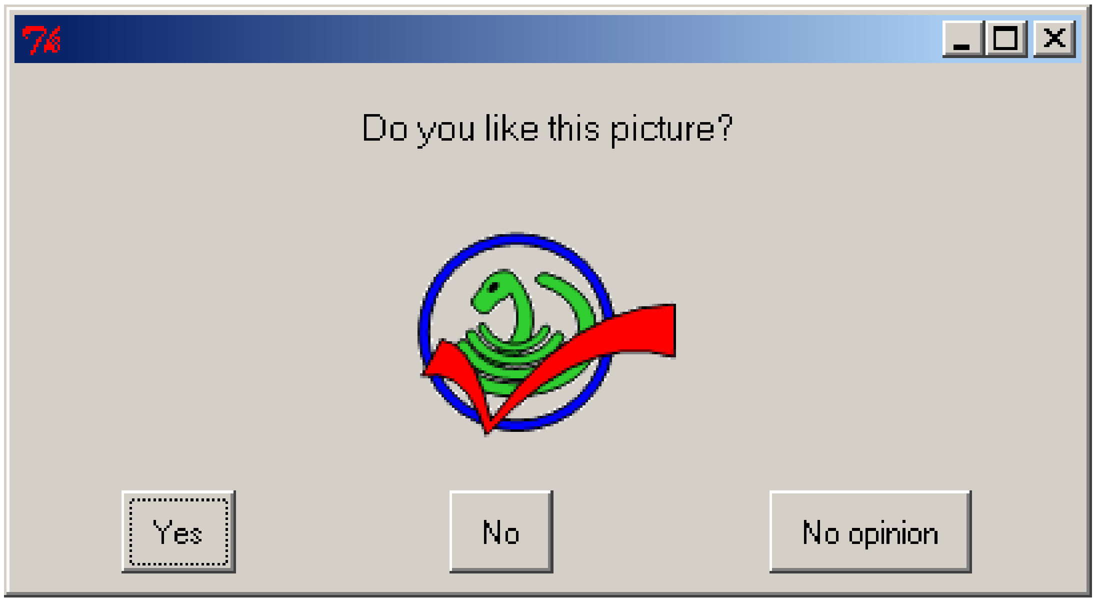
Note: The types of files supported depends on how you installed python. If other formats don't work, you may need to install the PIL library.

If an image argument is specified, the image file will be displayed after the message.

Here is some sample code from EasyGui's demonstration routine:

```
image = "python_and_check_logo.gif"
msg = "Do you like this picture?"
choices = ["Yes","No","No opinion"]
reply = buttonbox(msg, image=image, choices=choices)
```

If you click on one of the buttons on the bottom, its value will be returned in 'reply'. You may also click on the image. In that case, the image filename is returned.


Do you like this picture?

3.3.9 Letting the user select from a list of choices

choicebox

Buttonboxes are good for offering the user a small selection of short choices. But if there are many choices, or the text of the choices is long, then a better strategy is to present them as a list.

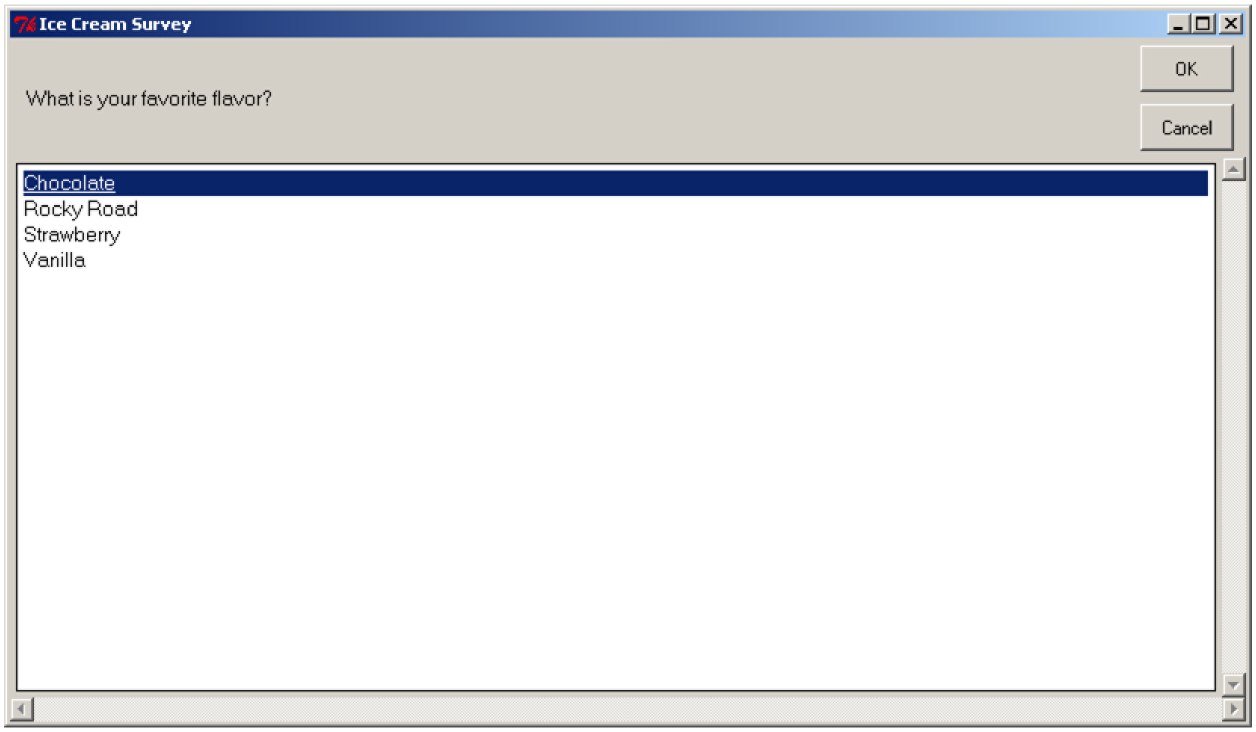
choicebox provides a way for a user to select from a list of choices. The choices are specified in a sequence (a tuple or a list). The choices will be given a case-insensitive sort before they are presented.

The keyboard can be used to select an element of the list.

Pressing "g" on the keyboard, for example, will jump the selection to the first element beginning with "g". Pressing "g" again, will jump the cursor to the next element beginning with "g". At the end of the elements beginning with "g", pressing "g" again will cause the selection to wrap around to the beginning of the list and jump to the first element beginning with "g".

If there is no element beginning with "g", then the last element that occurs before the position where "g" would occur is selected. If there is no element before "g", then the first element in the list is selected:

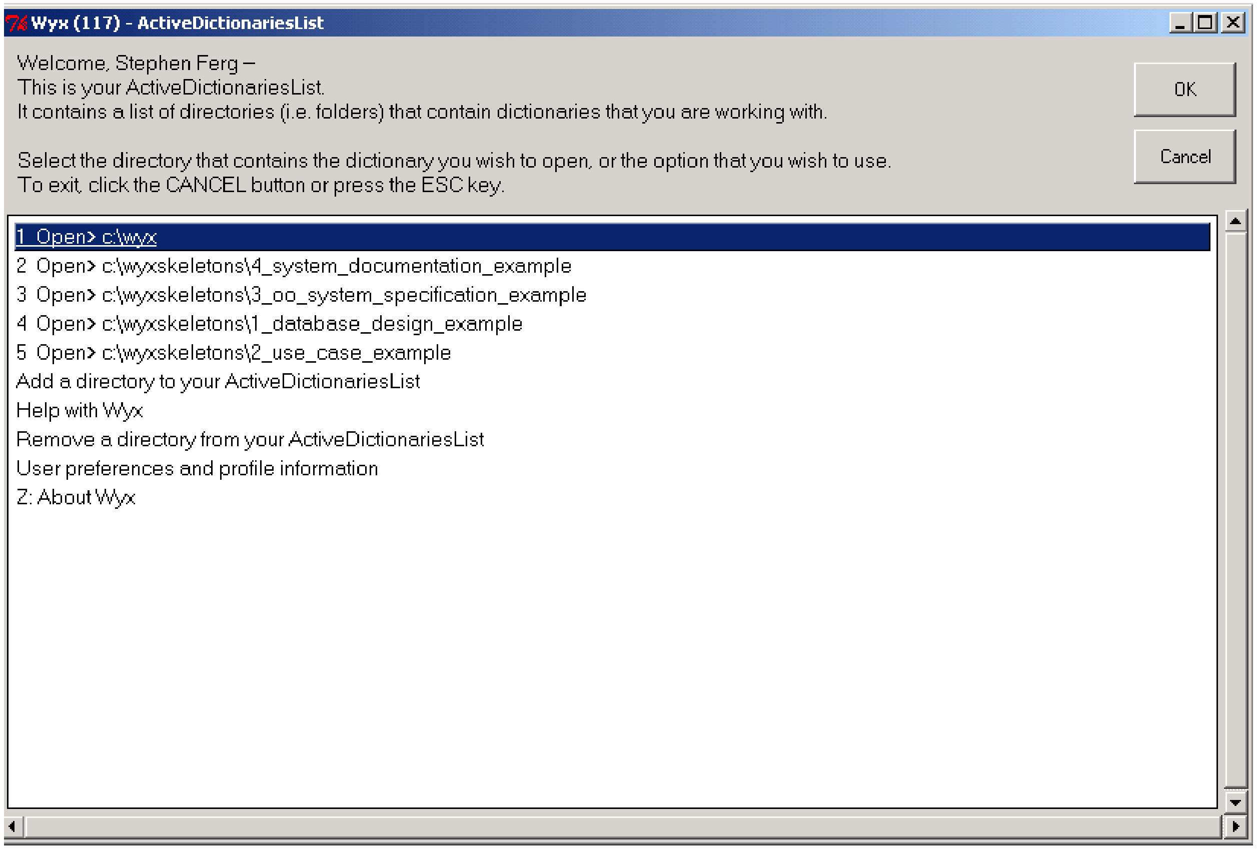
```
msg ="What is your favorite flavor?"
title = "Ice Cream Survey"
choices = ["Vanilla", "Chocolate", "Strawberry", "Rocky Road"]
choice = choicebox(msg, title, choices)
```



What is your favorite flavor?

Chocolate
Rocky Road
Strawberry
Vanilla

Another example of a choicebox:



Welcome, Stephen Ferg –
This is your ActiveDictionariesList.
It contains a list of directories (i.e. folders) that contain dictionaries that you are working with.

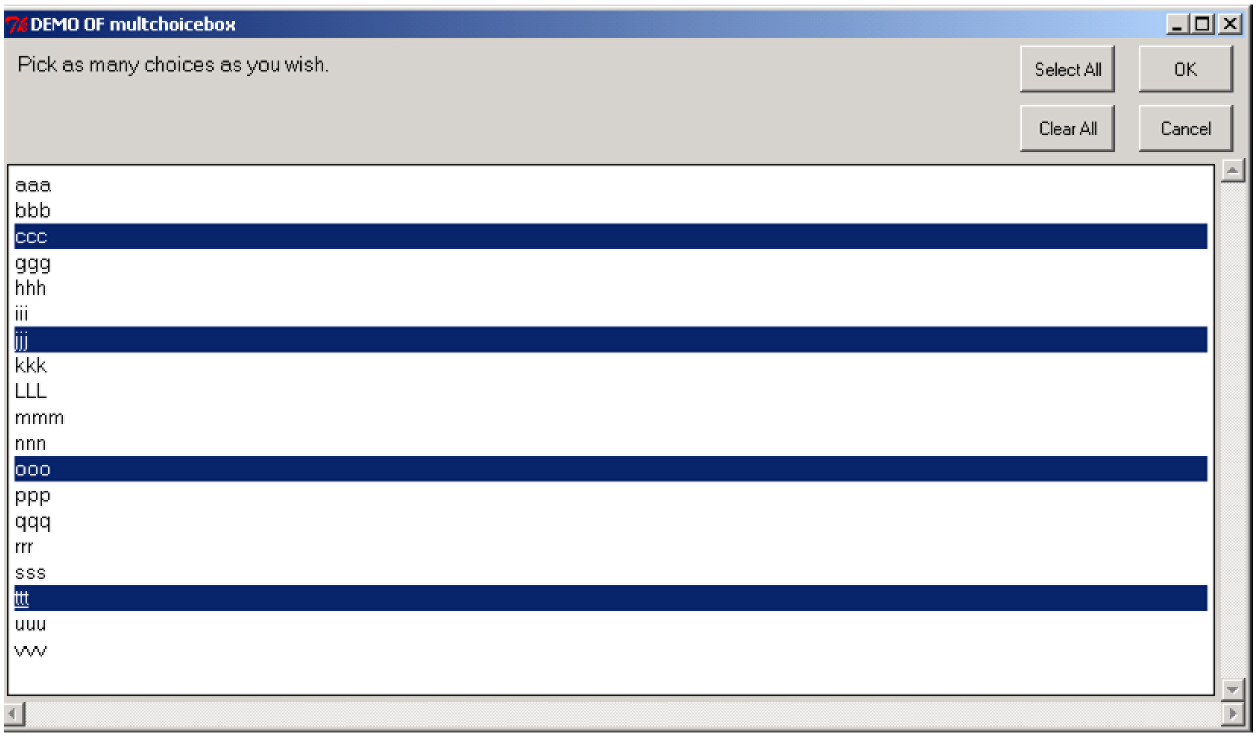
Select the directory that contains the dictionary you wish to open, or the option that you wish to use.
To exit, click the CANCEL button or press the ESC key.

1 Open> c:\wyx
2 Open> c:\wyxskeletons\4_system_documentation_example
3 Open> c:\wyxskeletons\3_oo_system_specification_example
4 Open> c:\wyxskeletons\1_database_design_example
5 Open> c:\wyxskeletons\2_use_case_example
Add a directory to your ActiveDictionariesList
Help with Wyx
Remove a directory from your ActiveDictionariesList
User preferences and profile information
Z: About Wyx

multchoicebox

The multchoicebox() function provides a way for a user to select from a list of choices. The interface looks just like the choicebox, but the user may select zero, one, or multiple choices.

The choices are specified in a sequence (a tuple or a list). The choices will be given a case-insensitive sort before they are presented.



Pick as many choices as you wish.

```
aaa
bbb
ccc
ggg
hhh
iii
jjj
kkk
LLL
mmm
nnn
ooo
ppp
qqq
rrr
sss
ttt
uuu
vvv
```

3.3.10 Letting the user enter information

enterbox

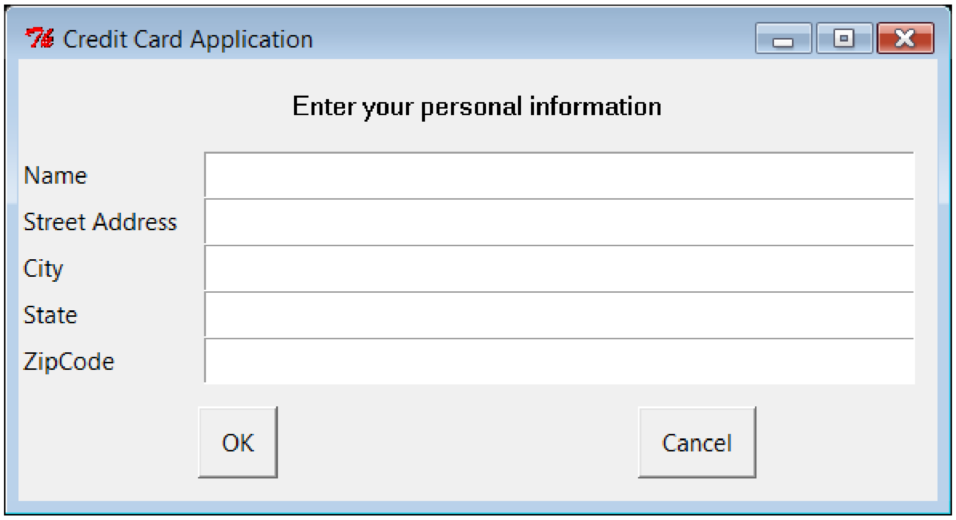
enterbox is a simple way of getting a string from the user

integerbox

integerbox is a simple way of getting an integer from the user.

multenterbox

multenterbox is a simple way of showing multiple enterboxes on a single screen.



In the multenterbox:

- If there are fewer values than names, the list of values is padded with empty strings until the number of values is the same as the number of names.
- If there are more values than names, the list of values is truncated so that there are as many values as names.

Returns a list of the values of the fields, or None if the user cancels the operation.

Here is some example code, that shows how values returned from multenterbox can be checked for validity before they are accepted:

```
from __future__ import print_function
msg = "Enter your personal information"
title = "Credit Card Application"
fieldNames = ["Name", "Street Address", "City", "State", "ZipCode"]
fieldValues = multenterbox(msg, title, fieldNames)
if fieldValues is None:
    sys.exit(0)
# make sure that none of the fields were left blank
while 1:
    errmsg = ""
    for i, name in enumerate(fieldNames):
        if fieldValues[i].strip() == "":
          errmsg += "{} is a required field.\n\n".format(name)
    if errmsg == "":
        break # no problems found
    fieldValues = multenterbox(errmsg, title, fieldNames, fieldValues)
    if fieldValues is None:
        break
print("Reply was:{}".format(fieldValues))
```

Note: The first line 'from __future__' is only necessary if you are using Python 2.*, and is only needed for this demo.

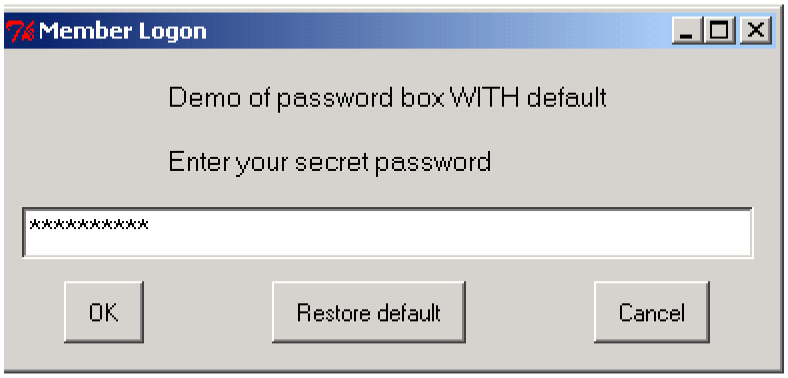
3.3.11 Letting the user enter password information

passwordbox

A passwordbox box is like an enterbox, but used for entering passwords. The text is masked as it is typed in.

multpasswordbox

multpasswordbox has the same interface as multenterbox, but when it is displayed, the last of the fields is assumed to be a password, and is masked with asterisks.



Demo of password box WITH default

Enter your secret password

****

3.3.12 Displaying text

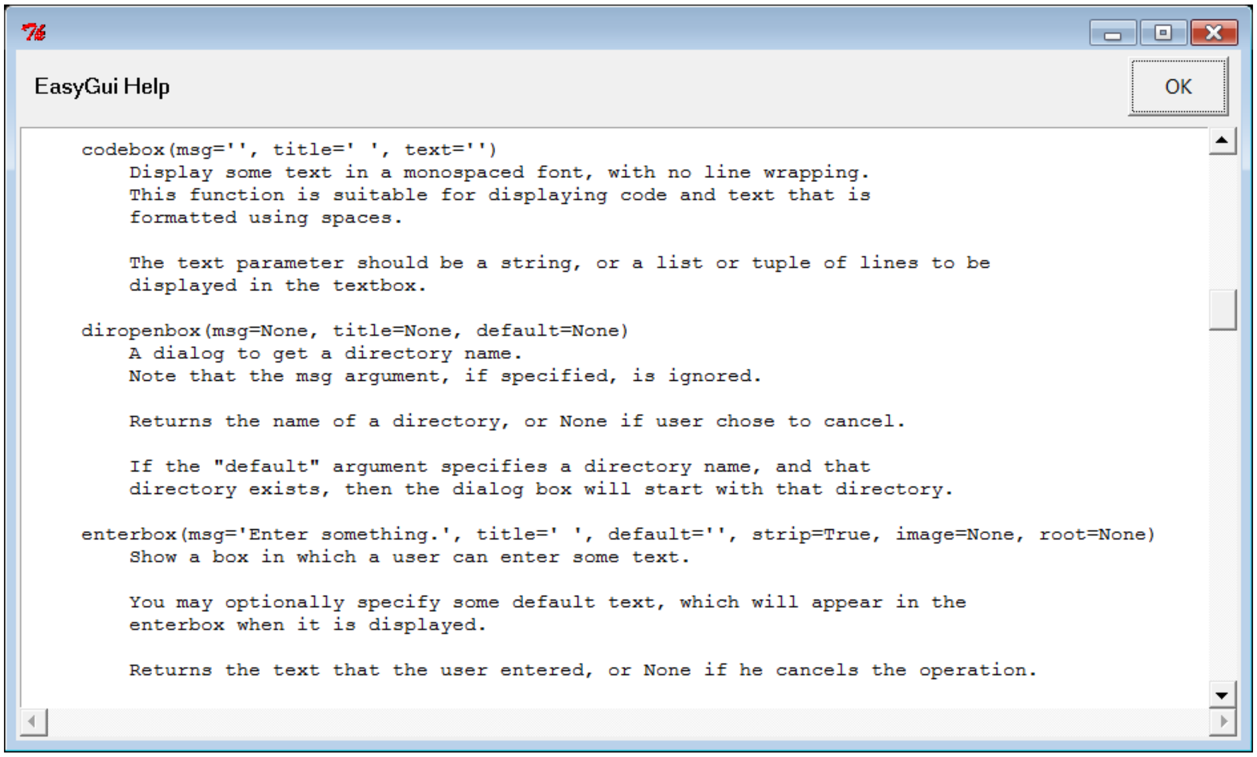
EasyGui provides functions for displaying text.

textbox

The textbox() function displays text in a proportional font. The text will word-wrap.

codebox

The codebox() function displays text in a monospaced font and does not wrap.



```
    codebox(msg='', title=' ', text='')
        Display some text in a monospaced font, with no line wrapping.
        This function is suitable for displaying code and text that is
        formatted using spaces.

        The text parameter should be a string, or a list or tuple of lines to be
        displayed in the textbox.

    diropenbox(msg=None, title=None, default=None)
        A dialog to get a directory name.
        Note that the msg argument, if specified, is ignored.

        Returns the name of a directory, or None if user chose to cancel.

        If the "default" argument specifies a directory name, and that
        directory exists, then the dialog box will start with that directory.

    enterbox(msg='Enter something.', title=' ', default='', strip=True, image=None, root=None)
        Show a box in which a user can enter some text.

        You may optionally specify some default text, which will appear in the
        enterbox when it is displayed.

        Returns the text that the user entered, or None if he cancels the operation.
```

Note that you can pass codebox() and textbox() either a string or a list of strings. A list of strings will be converted to text before being displayed. This means that you can use these functions to display the contents of a file this way:

```
import os
filename = os.path.normcase("c:/autoexec.bat")
f = open(filename, "r")
text = f.readlines()
f.close()
codebox("Contents of file " + filename, "Show File Contents", text)
```

3.3.13 Working with files

A common need is to ask the user for a filename or for a directory. EasyGui provides a few basic functions for allowing a user to navigate through the file system and choose a directory or a file. (These functions are wrappers around widgets and classes in lib-tk.)

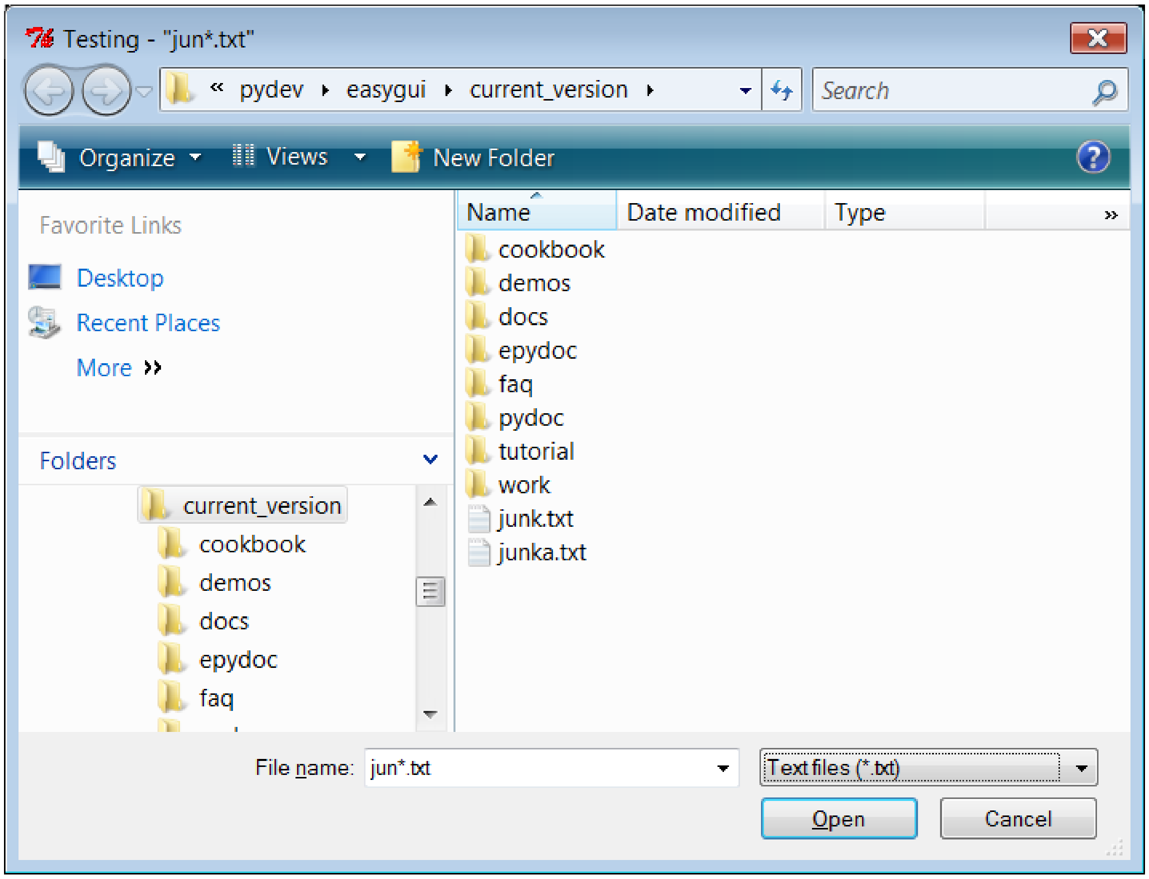
Note that in the current version of EasyGui, the startpos argument is not supported.

diropenbox

diropenbox returns the name of a directory

fileopenbox

fileopenbox returns the name of a file



filesavebox

filesavebox returns the name of a file

3.3.14 Remembering User Settings

EgStore

A common need is to ask the user for some setting, and then to "persist it", or store it on disk, so that the next time the user uses your application, you can remember his previous setting.

In order to make the process of storing and restoring user settings, EasyGui provides a class called EgStore. In order to remember some settings, your application must define a class (let's call it Settings, although you can call it anything you want) that inherits from EgStore.

Your application must also create an object of that class (let's call the object settings).

The constructor (the __init__ method) of the Settings class can initialize all of the values that you wish to remember.

Once you have done this, you can remember the settings simply by assigning values to instance variables in the settings object, and use the settings.store() method to persist the settings object to disk.

Here is an example of code using the Settings class:

```
from easygui import EgStore

# -----------------------------------------------------------------------
# define a class named Settings as a subclass of EgStore
# -------------------------------------------------------------------------
class Settings(EgStore):

    def __init__(self, filename):  # filename is required
        # -------------------------------------------------
        # Specify default/initial values for variables that
        # this particular application wants to remember.
        # -------------------------------------------------
        self.userId = ""
        self.targetServer = ""

        # -------------------------------------------------
        # For subclasses of EgStore, these must be
        # the last two statements in  __init__
        # -------------------------------------------------
        self.filename = filename  # this is required
        self.restore()

# Create the settings object.
# If the settingsFile exists, this will restore its values
# from the settingsFile.
# create "settings", a persistent Settings object
# Note that the "filename" argument is required.
# The directory for the persistent file must already exist.

settingsFilename = "settings.txt"
settings = Settings(settingsFilename)

# Now use the settings object.
# Initialize the "user" and "server" variables
# In a real application, we'd probably have the user enter them via enterbox
user    = "obama_barak"
server  = "whitehouse1"

# Save the variables as attributes of the "settings" object
settings.userId = user
settings.targetServer = server
settings.store()    # persist the settings
print("\nInitial settings")
print settings

# Run code that gets a new value for userId
# then persist the settings with the new value
user    = "biden_joe"
settings.userId = user
settings.store()
print("\nSettings after modification")
print settings

# Delete setting variable
del settings.userId
print("\nSettings after deletion of userId")
print settings
```

Here is an example of code using a dedicated function to create the Settings class:

```
from easygui import read_or_create_settings

# Create the settings object.
settings = read_or_create_settings('settings1.txt')

# Save the variables as attributes of the "settings" object
settings.userId = "obama_barak"
settings.targetServer = "whitehouse1"
settings.store()    # persist the settings
print("\nInitial settings")
print settings

# Run code that gets a new value for userId
# then persist the settings with the new value
user    = "biden_joe"
settings.userId = user
settings.store()
print("\nSettings after modification")
print settings

# Delete setting variable
del settings.userId
print("\nSettings after deletion of userId")
print settings
```

3.3.15 Trapping Exceptions

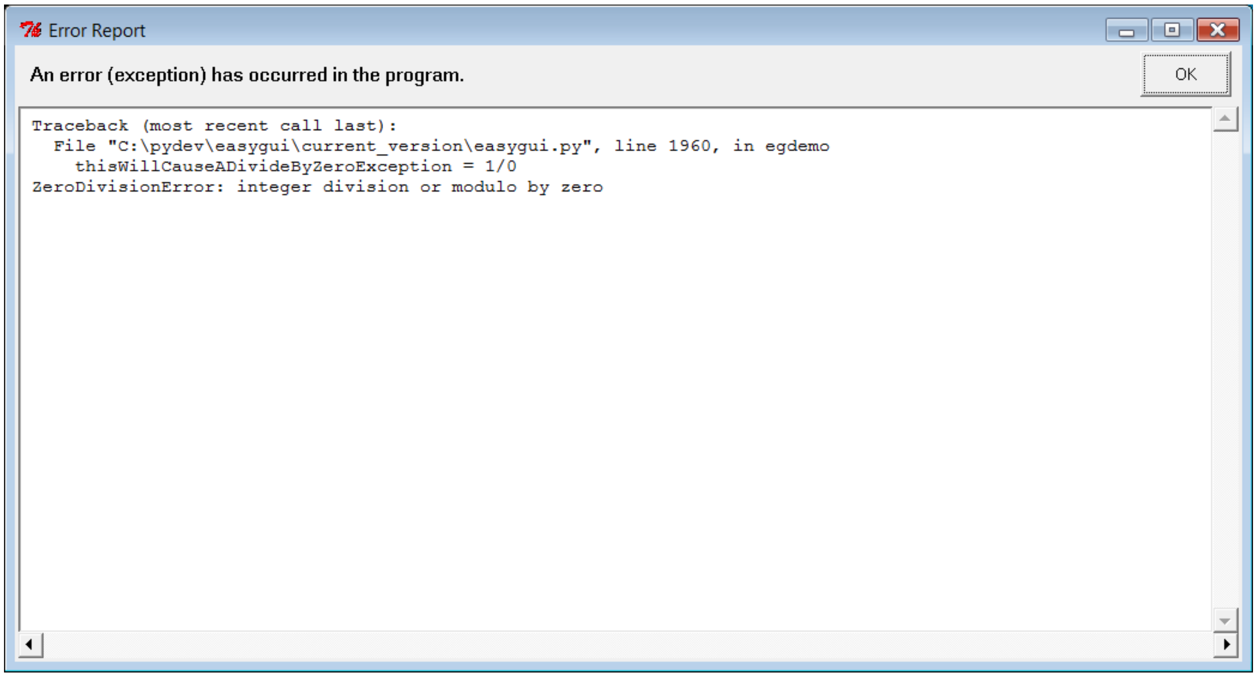
exceptionbox

Sometimes exceptions are raised... even in EasyGui applications. Depending on how you run your application, the stack trace might be thrown away, or written to stdout while your application crashes.

EasyGui provides a better way of handling exceptions via exceptionbox. Exceptionbox displays the stack trace in a codebox and may allow you to continue processing.

Exceptionbox is easy to use. Here is a code example:

```
try:
    someFunction()  # this may raise an exception
except:
    exceptionbox()
```



An error (exception) has occurred in the program.

```
Traceback (most recent call last):
  File "C:\pydev\easygui\current_version\easygui.py", line 1960, in egdemo
    thisWillCauseADivideByZeroException = 1/0
ZeroDivisionError: integer division or modulo by zero
```

3.4 EasyGui FAQ

3.4.1 An FAQ consisting of far too few questions. Help please :)

General Questions

1. What other gui libraries can I use?

   There are several. The two most popular as of 2014-12 are TkInter and PyQt.

   TkInter is a library shipped with Python and it is the de-facto standard for Python. You can find more about it at https://wiki.python.org/moin/TkInter

   PyQt is a very popular library. More information on it is at https://wiki.python.org/moin/PyQt

   A library inspired by easygui is the EasyGUI_qt project at http://easygui-qt.readthedocs.org/en/latest/ "Under the hood" easygui uses Tkinter while EasyGUI_qt uses pyQt

2. Why should I use easygui instead of some other library?

   Well, sometimes you should start with those other (excellent) libraries. However, we hope that you find easygui useful. Some of the cases for using easygui are:

   - You are starting to program and are tired of the command line >>>. easygui allows you to quickly create GUIs without worrying about all the details of Tk or Qt.
   - You already have a program and want to make it easier for people to use by building a GUI for it.
   - Its easy! You can try it out in a couple of hours and decide for yourself

   Don't worry. With easygui you are learning the basics. We take only a few shortcuts to make things easier. If you decide to move to a library with more functionality, you will already understand some of the GUI basics.

Specifics

1. Can I specify a custom sort order for the items in a choicebox?

   No, there is no way to specify a custom order. The reason is that the data must be sorted in order for the "jump to" feature (namely, to jump down in the list by pressing keyboard keys) to work.

   For some tips on how to control the order of items in a choicebox, see this recipe from the Easygui Cookbook.

2. Button box images don't appear and I get an error such as:

   > Cannot load C:UsersRobertSkyDriveGitHubeasyguieasyguipython_and_check_logo.jpg. Check to make sure it is an image file. PIL library isn't installed. If it isn't installed, only .gif files can be used.

   Possibly you are trying to load files other than .gif. Unfortunately, the 'PIL' library must be installed for this to work.

3.5 Cookbook

3.5.1 A section to hold code snippets and recipes

1. Simple demo program

   Here is a simple demo program using easygui. The screens that it produces are shown on the easygui home page.

```
from easygui import *
import sys

while 1:
    msgbox("Hello, world!")

    msg ="What is your favorite flavor?"
    title = "Ice Cream Survey"
    choices = ["Vanilla", "Chocolate", "Strawberry", "Rocky Road"]
    choice = choicebox(msg, title, choices)

    # note that we convert choice to string, in case
    # the user cancelled the choice, and we got None.
    msgbox("You chose: " + str(choice), "Survey Result")

    msg = "Do you want to continue?"
    title = "Please Confirm"
    if ccbox(msg, title):     # show a Continue/Cancel dialog
        pass  # user chose Continue
    else:
        sys.exit(0)           # user chose Cancel
```

2. Controlling the order of items in choicebox

   In a choicebox, the choices must be in sort order so that the keyboard "jump to" feature (jump down in the list by pressing keyboard keys) will work. But it often happens that a sort of first-cut listing of choices doesn't sort in a user-friendly order. So what can you do to control the order of the items displayed in a choicebox?

   A useful technique is to specify keys for the items in the choicebox. For example, suppose you want a choicebox to display View, Update, Delete, Exit. If you specified your choices this way:

```
choices = ["View", "Update", "Delete", "Exit"]
```

you'd get this:

- Delete
- Exit
- Update
- View

It is definitely in alphabetic order, but not very user-friendly. But if you specified keys for your choices this way:

```
choices = ["V View", "U Update", "D elete", "X Exit"]
```

you'd get this (with "X" appearing at the bottom):

- D Delete
- U Update
- V View
- X Exit

Suppose you wanted to force View to the top, so it is the easiest choice to select. You could change its key from "V" to "A":

```
choices = ["A View", "U Update", "D elete", "X Exit"]
```

and you'd get this:

- A View
- D Delete
- U Update
- X Exit

Another technique is to prepend a space to the choice. Since space characters always sorts before a non-space character, you can use this trick to force something like "V View" to the top of the list:

```
choices = [" V View", "U Update", "D Delete", "X Exit"]
```

produces this:

- V View
- D Delete
- U Update
- X Exit

In the proportional font used by choicebox, the space before the "V" is almost imperceptible.

Personally, I prefer to use alphabetic keys rather than numeric keys for choicebox items. It is easier to navigate the choices using alpha keys on the keyboard than by using the number keys.

And it is possible to use multi-character keys, like this:

- L1 Log old version
- L2 Log new version

Using keys for choices also makes it relatively easy to check for the user's selection:

```
    choices = [" V View", "U Update", "D elete", "X Exit"]
    choice = choicebox(msg,title,choices)

    if choice == None:
        return
    reply = choice.split()[0] # reply = the first word of the choice

    if reply == "X":
        return
    elif reply == "V":
        processView()
    elif reply == "L1":
        saveLog(version="old")
    elif reply == "L2":
        saveLog(version="new")
```

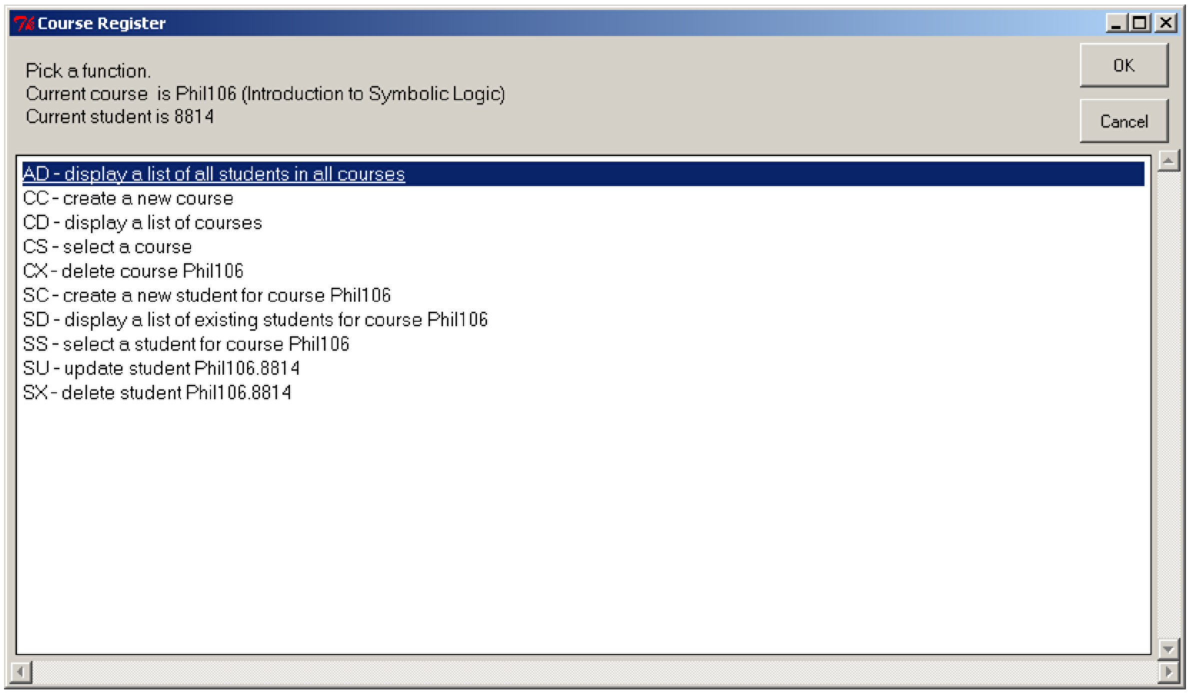
3. Registration System demo

   The Registration System demo application is a simple database application to maintain a list of courses, and students who are registered for the courses.

   It is not completely implemented – its purpose is to give you a feel for what is possible with EasyGui and how you might do it, not to be a complete working application.

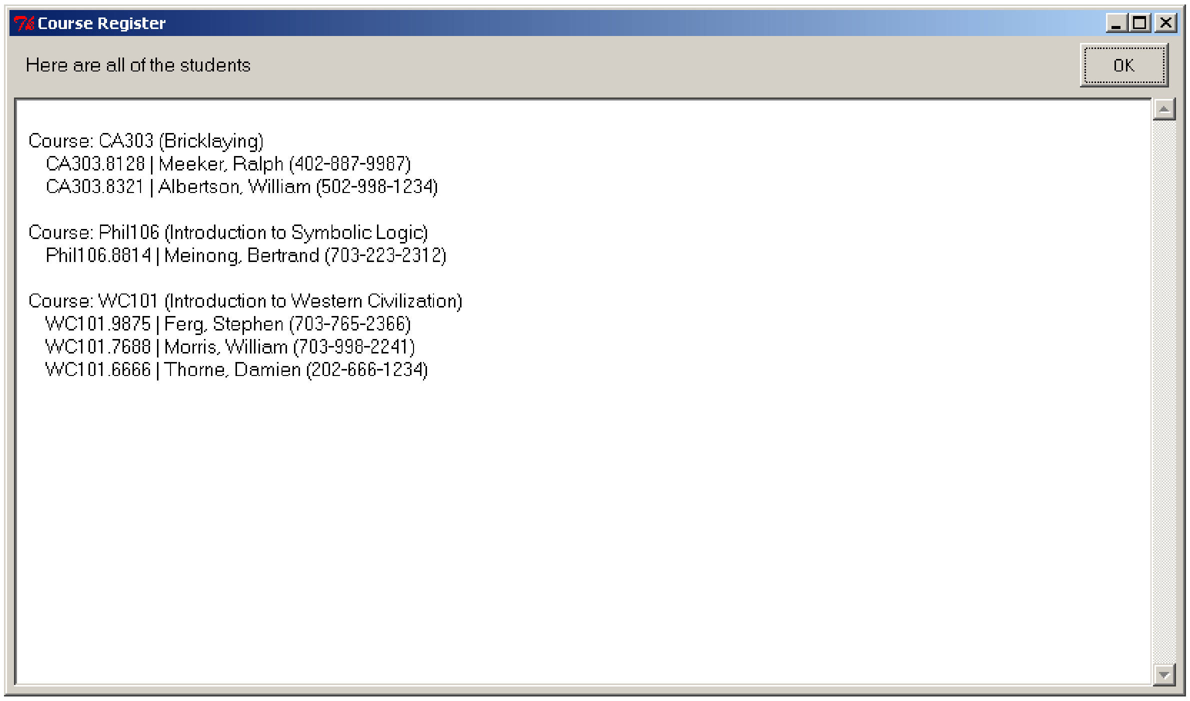
   File: `registration zip file`

   Screenshots:



Pick a function.
Current course is Phil106 (Introduction to Symbolic Logic)
Current student is 8814

AD - display a list of all students in all courses
CC - create a new course
CD - display a list of courses
CS - select a course
CX - delete course Phil106
SC - create a new student for course Phil106
SD - display a list of existing students for course Phil106
SS - select a student for course Phil106
SU - update student Phil106.8814
SX - delete student Phil106.8814



Here are all of the students

```
Course: CA303 (Bricklaying)
   CA303.8128 | Meeker, Ralph (402-887-9987)
   CA303.8321 | Albertson, William (502-998-1234)

Course: Phil106 (Introduction to Symbolic Logic)
   Phil106.8814 | Meinong, Bertrand (703-223-2312)

Course: WC101 (Introduction to Western Civilization)
   WC101.9875 | Ferg, Stephen (703-765-2366)
   WC101.7688 | Morris, William (703-998-2241)
   WC101.6666 | Thorne, Damien (202-666-1234)
```

3.6 EasyGui Links

–

3.6.1 YouTube videos by Marcus Adams

A nice series of tutorials were created to show off Message Box, Button Box, Continue Box and Choice:

Part 1 of many

3.6.2 easygui for Students

Austrian game-programming-for-children teacher Horst JENS says:

"I can proudly report that EasyGui is very popular with my students of all ages and one of the best ways to introduce young students to python in a short time."

Horst and his students have made several videos showing their use of EasyGui:

- Lexi shows msgbox
- Leo shows buttonbox
- Mira and Teresa show a simple graphic adventure game

3.6.3 Previous Versions and website

In December 2014, this website was converted to a new format. A snapshot of the older website is still available, but is static as it won't be updated: easygui website up to version 0.96

3.6.4 Stephen Ferg

Stephen developed easygui up through version 0.96. We are forever greatful for his insight and vision. He can be contacted at his website at: http://www.ferg.org/index.html

You might also visit his blog where much of easygui is discussed at: https://easygui.wordpress.com

Thank you Stephen.

- genindex
- search

Background

easygui was started several years ago by Stephen Ferg and was developed and supported by him through 2013. From there, work was restarted circa 2014. The first goal was to update the then four year old release and address some bugs and minor enhancements. That first release was 0.97

CHAPTER 4

LICENSE INFORMATION

4.1 ABOUT THE EASYGUI LICENSE

This license is what is generally known as the "modified BSD license",
aka "revised BSD", "new BSD", "3-clause BSD".
See http://opensource.org/licenses/bsd-license.php

This license is GPL-compatible.
See http://en.wikipedia.org/wiki/License_compatibility
See http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses

The BSD License is less restrictive than GPL.
It allows software released under the license to be incorporated into proprietary products.

Works based on the software may be released under a proprietary license or as closed source software.

http://en.wikipedia.org/wiki/BSD_licenses#3-clause_license_.28.22Revised_BSD_License.22.2C_.22New_BSD_License.22.2C_or_.22Modified_BSD_License.22.29

Python Module Index