Pyforms GUI Documentation

Release 4.9.2

Ricardo Jorge Vieira Ribeiro

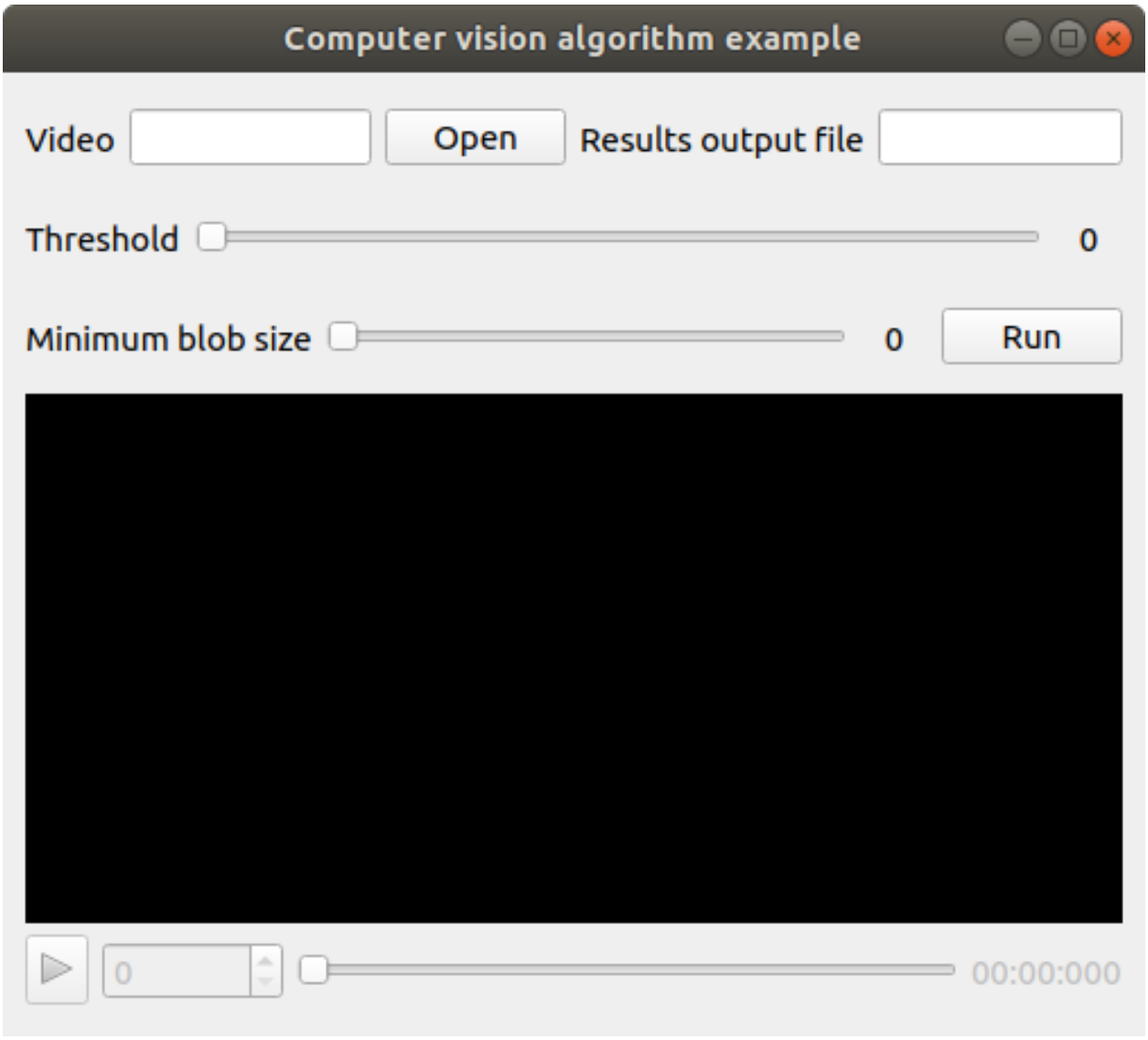
Jul 12, 2019

PYFORMS GUI

Pyforms GUI is Python 3 framework to allow pyforms applications to execute in Windows GUI mode.

The framework aims the boost the development productivity by providing an API in Python to allow the execution of applications developed for GUI and Web mode in terminal mode.

Source code https://github.com/UmSenhorQualquer/pyforms-gui



Note: This framework is a software layer part of the Pyforms framework.

Pyforms https://pyforms.readthedocs.io

CHAPTER

ONE

OVERVIEW

1.1 Pyforms GUI

Pyforms GUI is part the Pyforms framework. It implements a software layer that handles the execution of pyforms applications in Windows GUI mode.

1.2 Pyforms

Pyforms is a Python 3 framework to develop applications capable of executing in 3 diferent environments, Desktop GUI, Terminal and Web.

1.3 Developer

Ricardo Ribeiro	Champalimaud Scientific Software Platform ricardo.ribeiro@research.fchampalimaud.org ricardojvr@gmail.com

Note: Please **star** the project at the Github repository to support the project.

CHAPTER

TWO

INSTALL & CONFIGURE

- Install Pyforms using **pip**.

```
pip install pyforms-gui
```

CHAPTER

THREE

FIRST APPLICATION

Note: More documentation to read about this example at:

- *pyforms_gui.basewidget.BaseWidget*
- *pyforms_gui.controls.control_base.ControlBase*

Here it is shown how to create the first pyforms app.

3.1 Create the first app

Create the file **example.py** and add the next code to it.

```
from pyforms.basewidget import BaseWidget
from pyforms.controls   import ControlFile
from pyforms.controls   import ControlText
from pyforms.controls   import ControlSlider
from pyforms.controls   import ControlPlayer
from pyforms.controls   import ControlButton

class ComputerVisionAlgorithm(BaseWidget):

    def __init__(self, *args, **kwargs):
        super().__init__('Computer vision algorithm example')

        #Definition of the forms fields
        self._videofile     = ControlFile('Video')
        self._outputfile    = ControlText('Results output file')
        self._threshold     = ControlSlider('Threshold', default=114, minimum=0,␣
↪maximum=255)
        self._blobsize      = ControlSlider('Minimum blob size', default=110,␣
↪minimum=100, maximum=2000)
        self._player        = ControlPlayer('Player')
        self._runbutton     = ControlButton('Run')

        #Define the function that will be called when a file is selected
        self._videofile.changed_event     = self.__videoFileSelectionEvent
        #Define the event that will be called when the run button is processed
        self._runbutton.value       = self.__runEvent
        #Define the event called before showing the image in the player
        self._player.process_frame_event    = self.__process_frame
```

(continues on next page)

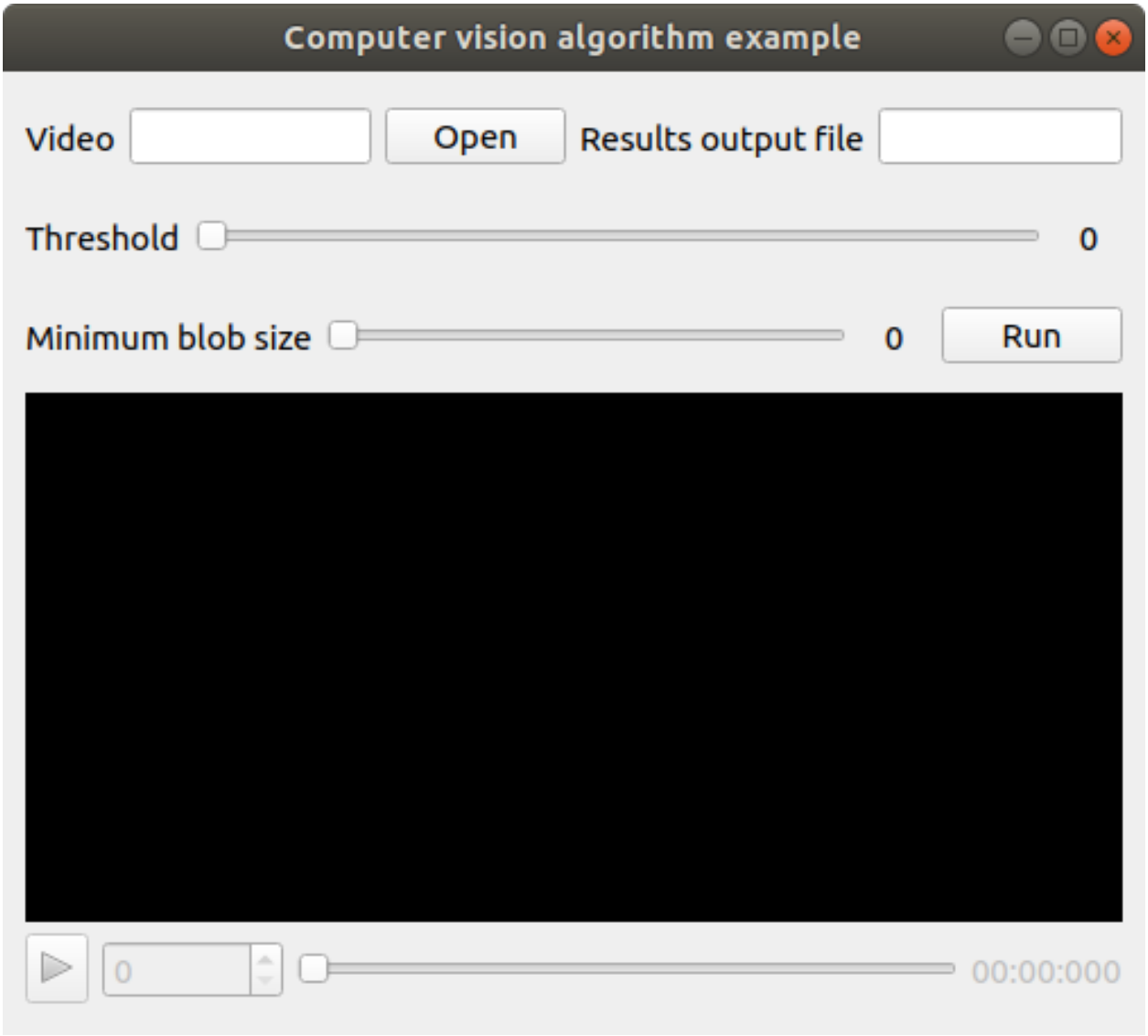
(continued from previous page)

```
        #Define the organization of the Form Controls
        self._formset = [
            ('_videofile', '_outputfile'),
            '_threshold',
            ('_blobsize', '_runbutton'),
            '_player'
        ]


    def __videoFileSelectionEvent(self):
        """
        When the videofile is selected instanciate the video in the player
        """
        self._player.value = self._videofile.value

    def __process_frame(self, frame):
        """
        Do some processing to the frame and return the result frame
        """
        return frame

    def __runEvent(self):
        """
        After setting the best parameters run the full algorithm
        """
        pass


if __name__ == '__main__':

    from pyforms import start_app
    start_app(ComputerVisionAlgorithm)
```

Now execute in the terminal the next command:

```
$ python example.py
```

You will visualize the next result:


Computer vision algorithm example
Video
Open
Results output file
Threshold
0
Minimum blob size
0
Run
0
00:00:000

CHAPTER

FOUR

THE BASICS

This page was based on the examples available at the github folder: Tutorial - SimpleExamples

4.1 Prepare the application class

Create the Python file that will store your applications.

Example: **SimpleExample.py**

4.1.1 Import the library

Import the pyforms library, the BaseWidget and the Controls classes that you will need:

```
import pyforms
from   pyforms.basewidget import BaseWidget
from   pyforms.controls import ControlText
from   pyforms.controls import ControlButton
```

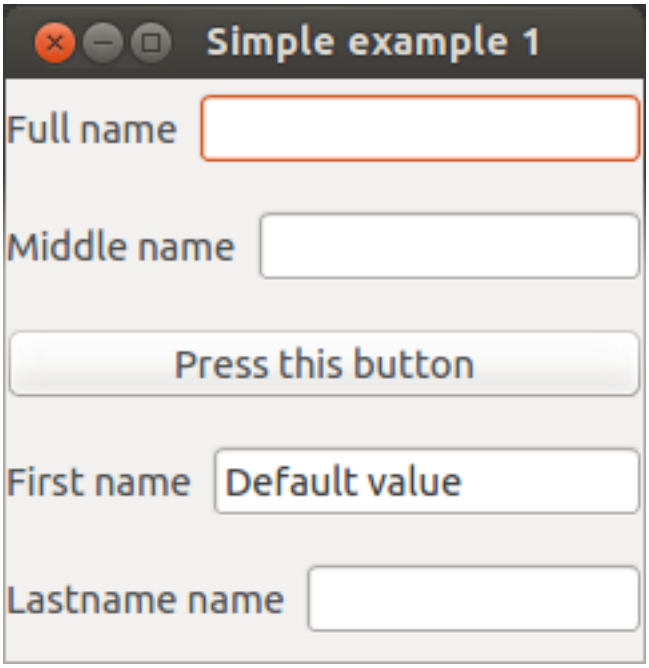
4.1.2 Create your application class

This class should inherit from the class BaseWidget.

```
class SimpleExample1(BaseWidget):

    def __init__(self):
        super(SimpleExample1,self).__init__('Simple example 1')

        #Definition of the forms fields
        self._firstname  = ControlText('First name', 'Default value')
        self._middlename = ControlText('Middle name')
        self._lastname   = ControlText('Lastname name')
        self._fullname   = ControlText('Full name')
        self._button     = ControlButton('Press this button')

#Execute the application
if __name__ == "__main__": pyforms.start_app( SimpleExample1 )
```

If you run this file, it will produce the next window.

SimpleExample1



4.2 Add an action to the button

4.2.1 Create the action

Create the class function that will work as the button action.

```
def __buttonAction(self):
    """Button action event"""
    self._fullname.value = self._firstname.value +" "+ self._middlename.value +"
↪"+self._lastname.value
```

4.2.2 Set the button action

Configure the button to execute your function when pressed. Inside the class constructor add the code:

```
#Define the button action
self._button.value = self.__buttonAction
```

The final code should look like:

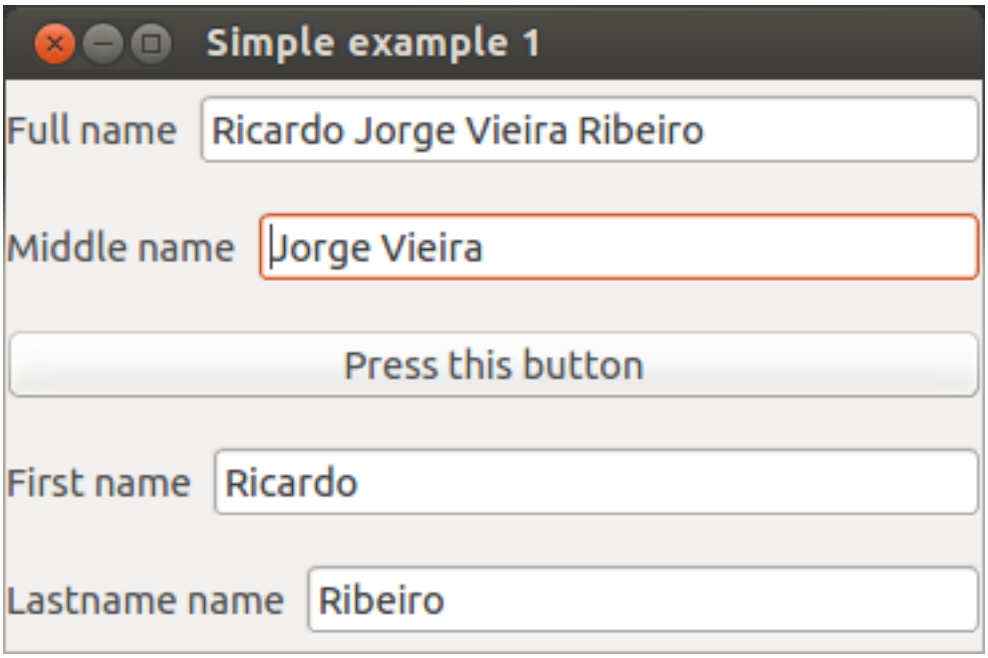
```
import pyforms
from  pyforms.basewidget import BaseWidget
from  pyforms.controls import ControlText
from  pyforms.controls import ControlButton

class SimpleExample1(BaseWidget):

    def __init__(self):
        super(SimpleExample1,self).__init__('Simple example 1')
```

(continues on next page)

(continued from previous page)

```
        #Definition of the forms fields
        self._firstname  = ControlText('First name', 'Default value')
        self._middlename = ControlText('Middle name')
        self._lastname   = ControlText('Lastname name')
        self._fullname   = ControlText('Full name')
        self._button     = ControlButton('Press this button')

        #Define the button action
        self._button.value = self.__buttonAction

    def __buttonAction(self):
        """Button action event"""
        self._fullname.value = self._firstname.value +" "+ self._middlename.value + \
        " "+ self._lastname.value

#Execute the application
if __name__ == "__main__": pyforms.start_app( SimpleExample1 )
```

The previous code produces the next window, after you had pressed the button:



4.3 Organize your form Controls

Use the BaseWidget.formset variable to organize the Controls inside the Window. Find here more details about the formset variable

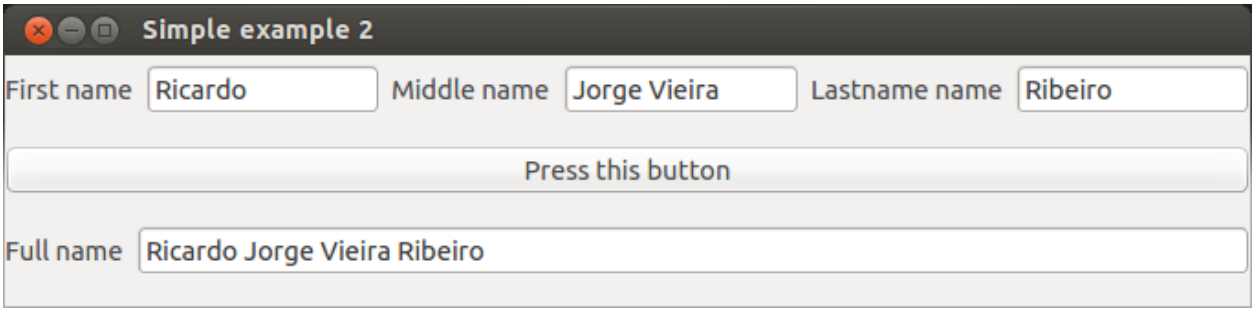
```
...

class SimpleExample1(BaseWidget):

    def __init__(self):
        ...
```

(continues on next page)

(continued from previous page)

```
        #Define the organization of the forms
        self.formset = [ ('_firstname','_middlename','_lastname'), '_button', '_
↪fullname', ' ']
        #The ' ' is used to indicate that a empty space should be placed at the␣
↪bottom of the window
        #If you remove the ' ' the forms will occupy the entire window

    ...
```

Result:



Try now:

```
self.formset = [ {
        'Tab1':['_firstname','||','_middlename','||','_lastname'],
        'Tab2':['_fullname']
    },
    '=',
    (' ','_button', ' ')
]
#Use dictionaries for tabs
#Use the sign '=' for a vertical splitter
#Use the signs '||' for a horizontal splitter
```

Note: In the name of each tab use the format **a:Tab1** and **b:Tab2** to define the order of the tabs. Example:

```
self.formset = [ {
    'a:Tab1':['_firstname','||','_middlename','||','_lastname'],
    'b:Tab2':['_fullname']
}
```

4.4 Add a main menu

To add a main menu to your application, first you need to define the functions that will work as the options actions.

```
...

class SimpleExample1(BaseWidget):
    ...
```

(continues on next page)

(continued from previous page)

```
    def __openEvent(self):
        ...

    def __saveEvent(self):
        ...

    def __editEvent(self):
        ...

    def __pastEvent(self):
        ...
```

After you just need to set the BaseWidget.mainmenu property inside your application class constructor as the example bellow.

```
...

class SimpleExample1(BaseWidget):

    def __init__(self):
        ...
        self.mainmenu = [
            { 'File': [
                    {'Open': self.__openEvent},
                    '-',
                    {'Save': self.__saveEvent},
                    {'Save as': self.__saveAsEvent}
                ]
            },
            { 'Edit': [
                    {'Copy': self.__editEvent},
                    {'Past': self.__pastEvent}
                ]
            }
        ]

    ...
```

4.5 Add popup menu to the Controls

Create the functions that will work as the popup menu options actions, as you have than in the main menu chapter. After use the functions **add_popup_menu_option** or **add_popup_sub_menu_option** to add a popup menu or a popup submenu to your Control.

[Find here more details about the functions add_popup_menu_option and add_popup_sub_menu_option.](http://pyforms.readthedocs.org/en/latest/api-documentation/controls/#controlbase)

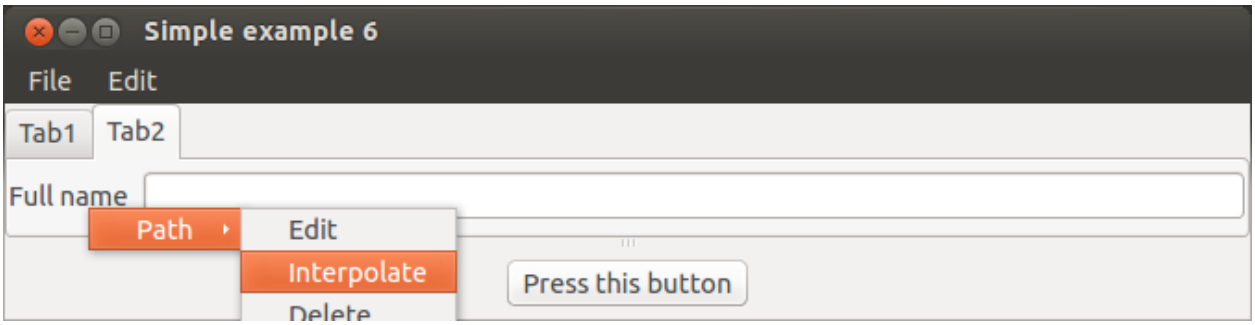
```
...

class SimpleExample1(BaseWidget):

    def __init__(self):
        ...
```

(continues on next page)

(continued from previous page)

```
        self._fullname.addPopupSubMenuOption('Path',
            {
                'Delete':           self.__dummyEvent,
                'Edit':             self.__dummyEvent,
                'Interpolate':      self.__dummyEvent
            })
    ...
```

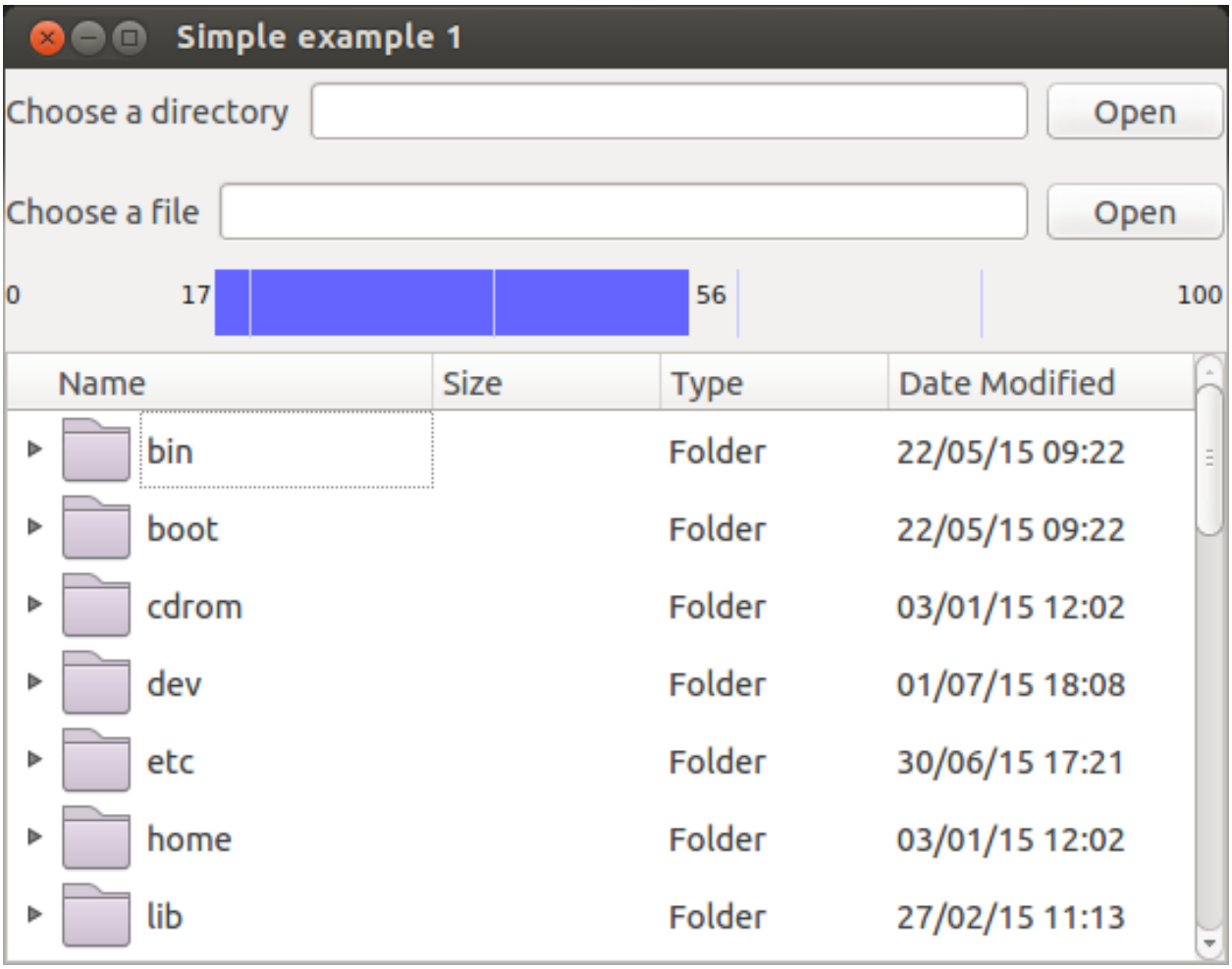
Result:

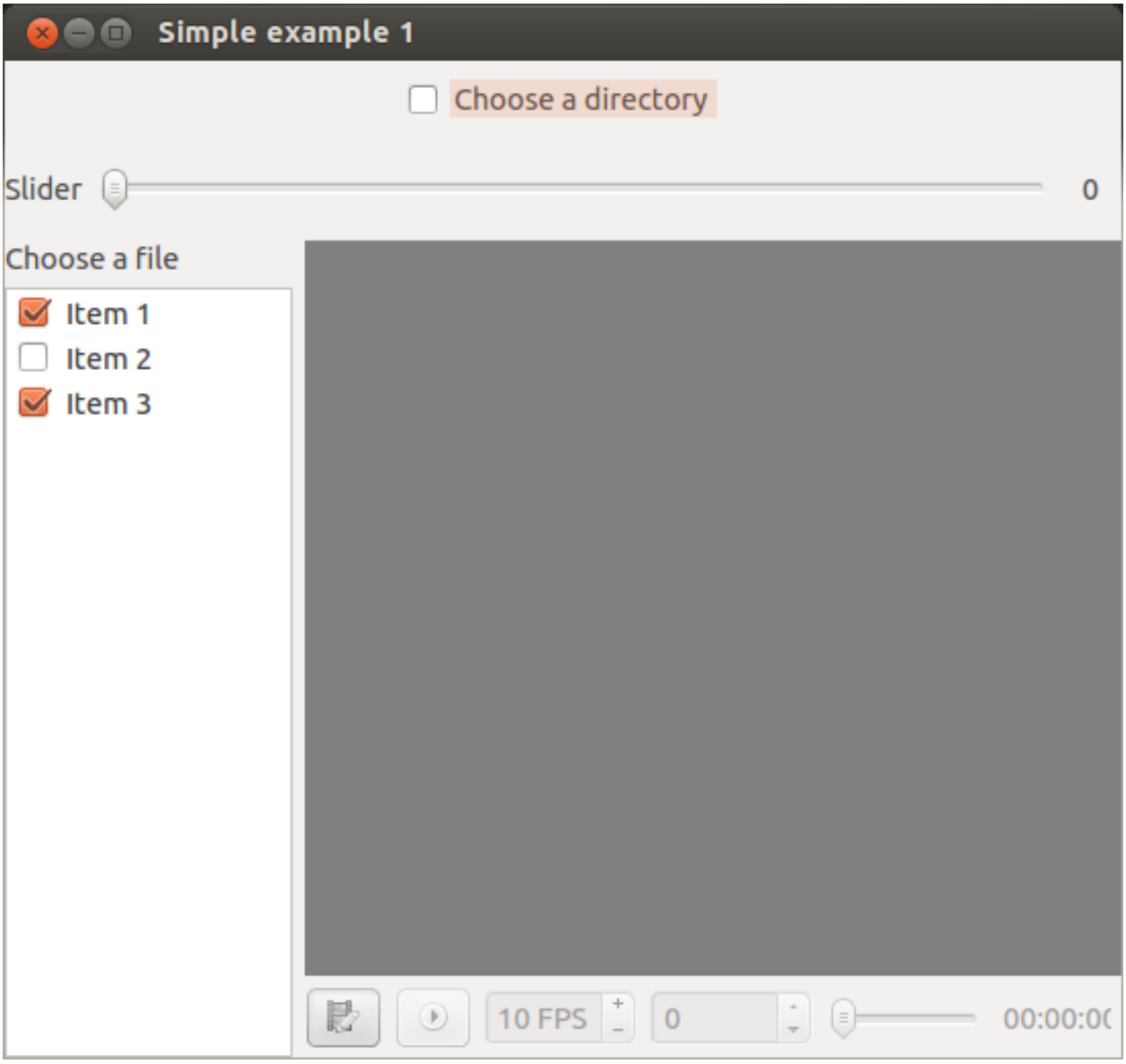


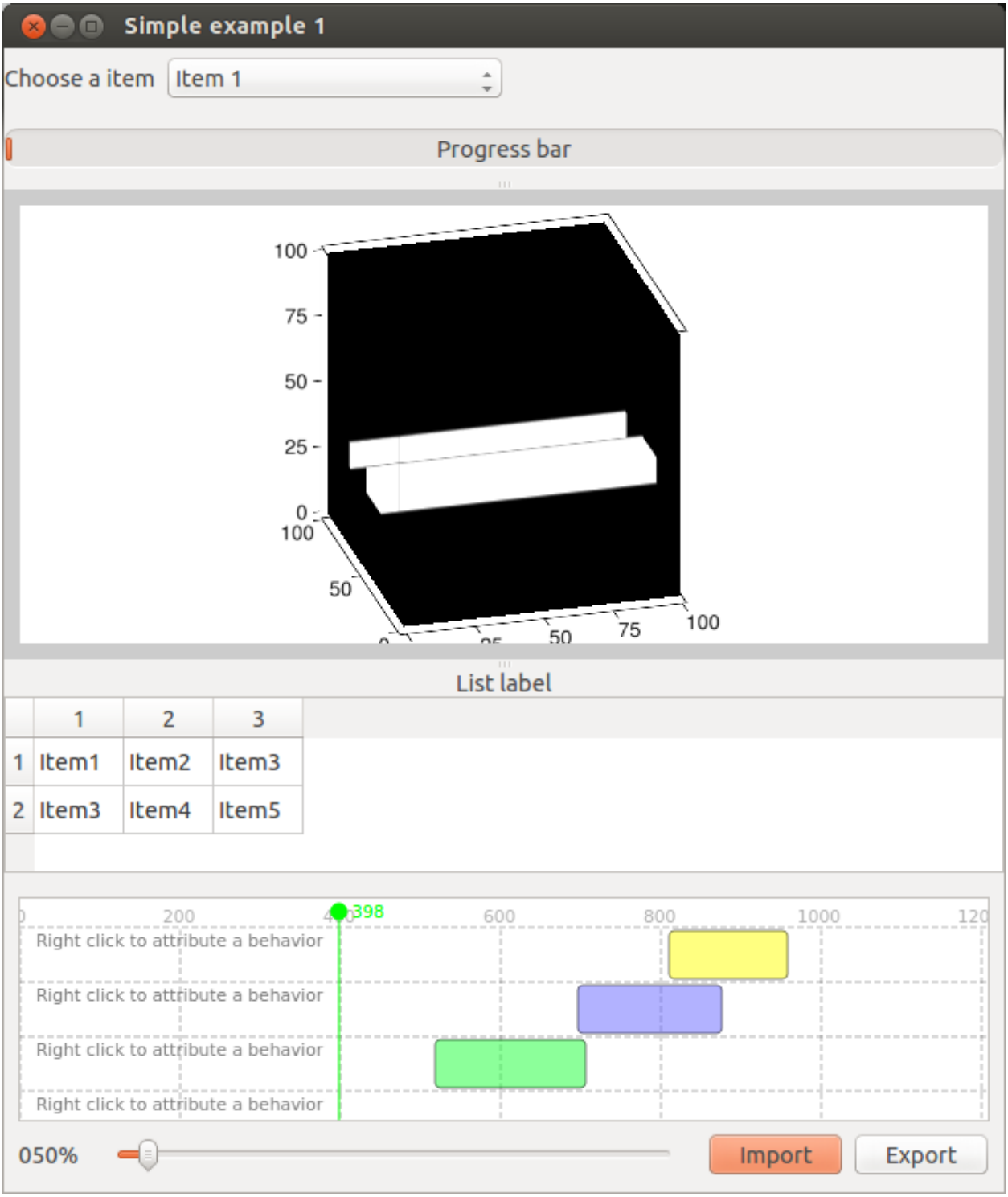
4.6 What's next?

4.6.1 Move to the next chapter.

4.6.2 Find out what you can do with other Controls here.




Simple example 1
Choose a directory
Slider
0
Choose a file
Item 1
Item 2
Item 3
10 FPS
0
00:00:0(


Simple example 1
Choose a item
Item 1
Progress bar
100
75
50
25
0
100
50
50
75
100
List label
1
2
3
1
Item1
Item2
Item3
2
Item3
Item4
Item5
200
398
600
800
1000
Right click to attribute a behavior
Right click to attribute a behavior
Right click to attribute a behavior
Right click to attribute a behavior
050%
Import
Export

CHAPTER

FIVE

MULTIPLE WINDOWS

This page was based on the examples available at the github folder: Tutorial - Code Organization

The application described on this page will allow us to add People details to a list.

5.1 Create the Model

Instead of starting by showing you how to develop the GUI I will suggest first how to modularize the code in a Model View Control (MVC) style.

First we will create our data model which may be used outside the GUI.

5.1.1 Data model

Start by creating the file Person.py where we will implement the model responsible for storing the a person information.

```
class Person(object):

    def __init__(self, firstName, middleName, lastName):
        self._firstName     = firstName
        self._middleName    = middleName
        self._lastName      = lastName

    @property
    def fullName(self):
        return "{0} {1} {2}".format(self._firstName, self._middleName, self._lastName)
```

After, create the file People.py and implement the People class which will keep and manage the list of people.

```
import pickle

class People(object):

    def __init__(self):
        self._people = []

    def addPerson(self, person):
        self._people.append(person)

    def removePerson(self, index):
        return self._people.pop(index)
```

(continues on next page)

(continued from previous page)

```
    def save(self, filename):
        output = open(filename, 'wb')
        pickle.dump(self._people, output)

    def load(self, filename):
        pkl_file = open(filename, 'rb')
        self._people = pickle.load(pkl_file)
```

5.1.2 Let's go for the GUI

To make our code modular and easy to navigate we will split the edition of the 2 Models in 2 different windows.

Implement the GUI to manage the Person Model.

Create the file PersonWindow.py and implement the window that will allow us the edit the Person Model. This window should inherit from the BaseWidget and Person classes.

```
import pyforms
from pyforms.basewidget import BaseWidget
from pyforms.controls   import ControlText
from pyforms.controls   import ControlButton
from Person             import Person

class PersonWindow(Person, BaseWidget):

    def __init__(self):
        Person.__init__(self, '', '', '')
        BaseWidget.__init__(self,'Person window')

        #Definition of the forms fields
        self._firstnameField    = ControlText('First name')
        self._middlenameField   = ControlText('Middle name')
        self._lastnameField     = ControlText('Lastname name')
        self._fullnameField     = ControlText('Full name')
        self._buttonField       = ControlButton('Press this button')

        #Define the button action
        self._buttonField.value = self.__buttonAction


    def __buttonAction(self):
        self._firstName             = self._firstnameField.value
        self._middleName            = self._middlenameField.value
        self._lastName              = self._lastnameField.value
        self._fullnameField.value   = self.fullName

        #In case the window has a parent
        if self.parent!=None: self.parent.addPerson(self)

    #Execute the application
    if __name__ == "__main__":   pyforms.start_app( PersonWindow )
```

Note: *Test the window by executing the file.*

5.1.3 Implement the GUI to manage the People model

Create the file PeopleWindow.py and implement the window that will allow us the manager the People Model. This window should inherit from the BaseWidget and People classes.

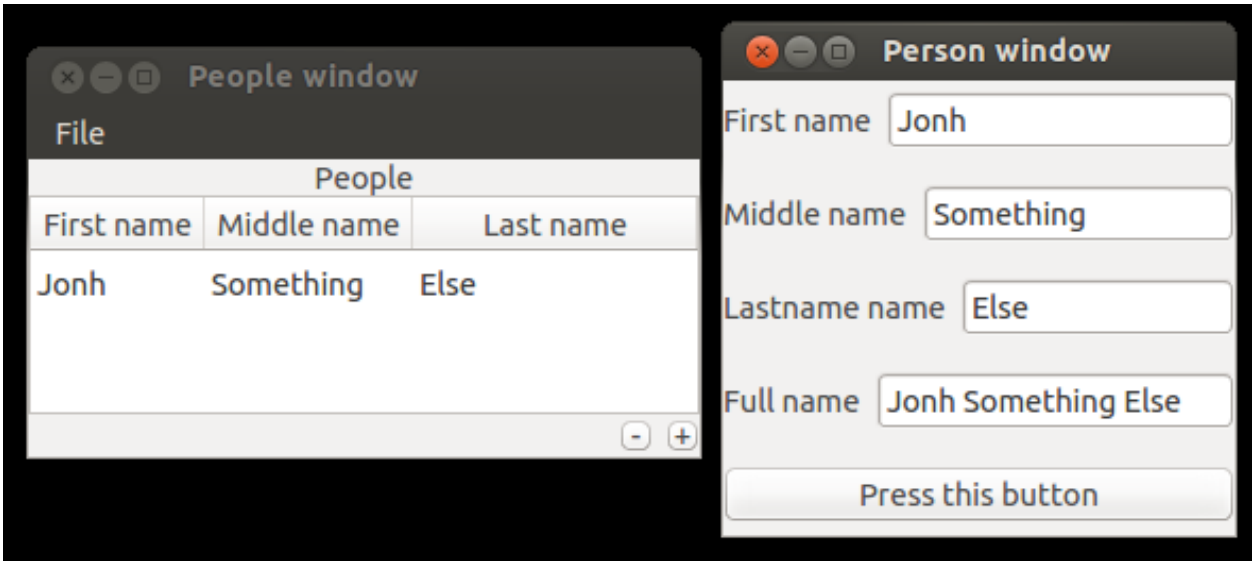
```
import pyforms
from pyforms.basewidget     import BaseWidget
from pyforms.controls       import ControlList
from People                 import People
from PersonWindow           import PersonWindow
from AddMenuFuntionality    import AddMenuFuntionality

class PeopleWindow(AddMenuFuntionality, People, BaseWidget):
    """
    This applications is a GUI implementation of the People class
    """

    def __init__(self):
        People.__init__(self)
        BaseWidget.__init__(self,'People window')

        #Definition of the forms fields
        self._peopleList    = ControlList('People',
            plusFunction    = self.__addPersonBtnAction,
            minusFunction   = self.__rmPersonBtnAction)

        self._peopleList.horizontalHeaders = ['First name', 'Middle name', 'Last name
↪']

    def addPerson(self, person):
        """
        Reimplement the addPerson function from People class to update the GUI
        everytime a new person is added.
        """
        super(PeopleWindow, self).addPerson(person)
        self._peopleList += [person._firstName, person._middleName, person._lastName]
        person.close() #After adding the person close the window

    def removePerson(self, index):
        """
        Reimplement the removePerson function from People class to update the GUI
        everytime a person is removed.
        """
        super(PeopleWindow, self).removePerson(index)
        self._peopleList -= index

    def __addPersonBtnAction(self):
        """
        Add person button event.
        """
        # A new instance of the PersonWindow is opened and shown to the user.
        win = PersonWindow()
        win.parent = self
        win.show()

    def __rmPersonBtnAction(self):
        """
        Remove person button event
```

(continues on next page)

(continued from previous page)

```
        """
        self.removePerson( self._peopleList.selected_row_index )

#Execute the application
if __name__ == "__main__":   pyforms.start_app( PeopleWindow )
```

The application will look like:



5.2 EmptyWidget Control

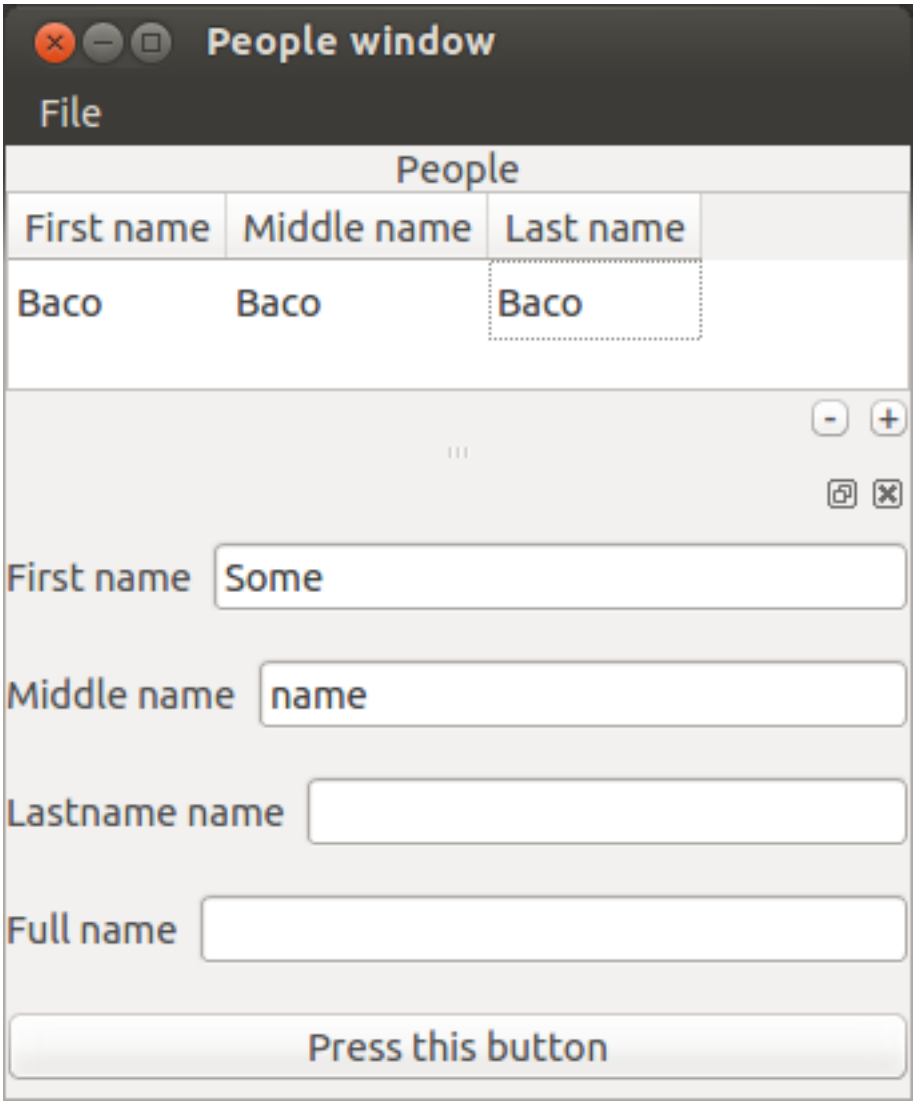
Instead of opening a new window everytime we want to add a new Person, we will change the Application to open the PersonWindow inside the PeopleWindow. For this we will use the ControlEmptyWidget.

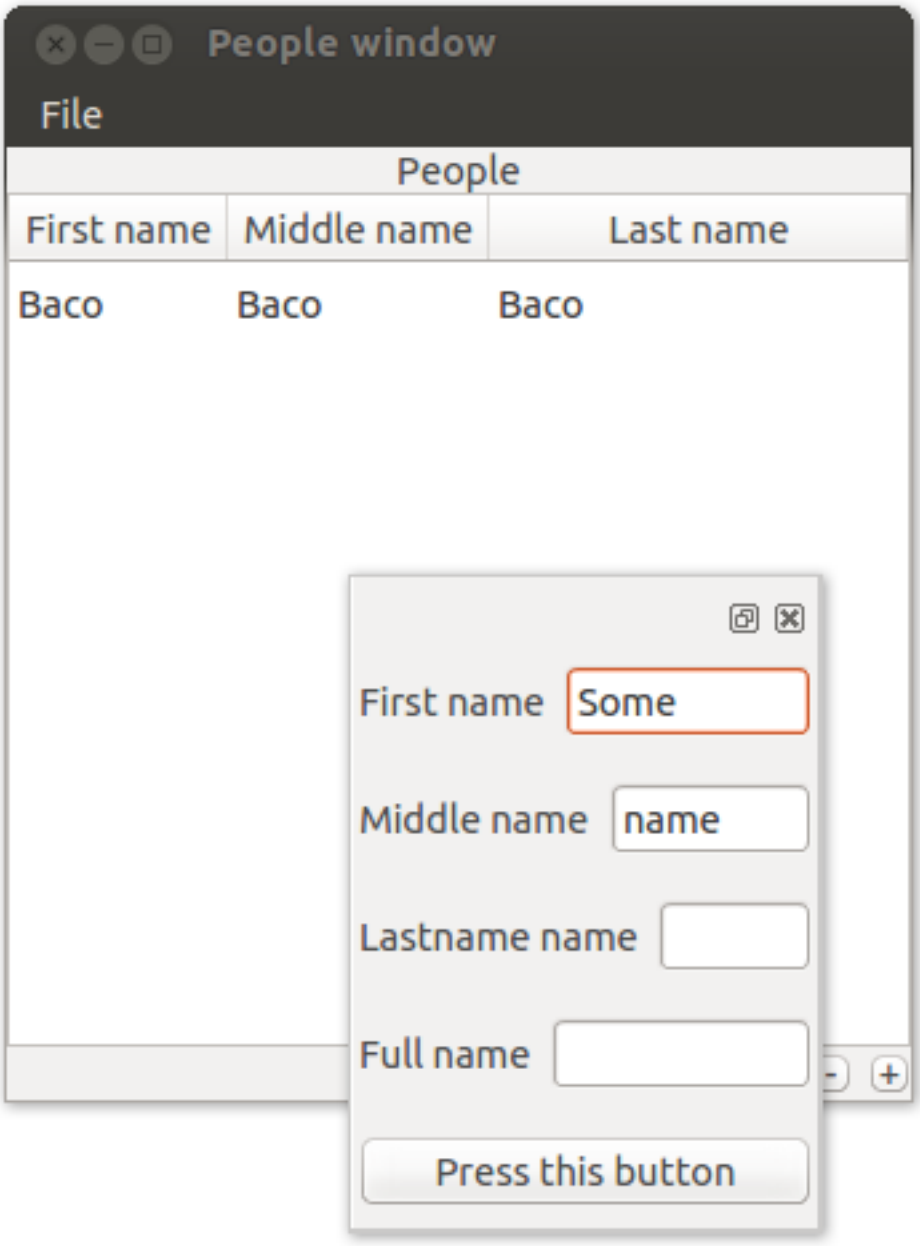
```
from pyforms.controls       import ControlEmptyWidget
...

    def __init__(self):
        ...
        self._panel = ControlEmptyWidget()

    def __addPersonBtnAction(self):
        """
        Add person button event.
        """
        # A new instance of the PersonWindow is opened and shown to the user.
        win = PersonWindow()
        win.parent = self
        self._panel.value = win

...
```

5.3 DockWidget Control

A DockWidget works like the EmptyWidget but can be detached or moved around the sides of the main Window.

```
from pyforms.controls import ControlDockWidget
...

    def __init__(self):
        ...
        self._panel = ControlDockWidget()
...
```




People window
File
People
First name
Middle name
Last name
Baco
Baco
Baco
First name
Some
Middle name
name
Lastname name
Full name
Press this button

CHAPTER

SIX

MDI APPLICATIONS

This page was based in the examples available on the github folder: Tutorial - Mdi Application

CHAPTER

SEVEN

STYLE AND LAYOUT WITH CSS

This page was based on the examples available at the github folder: Tutorial - Code Organization

PyForms takes advantage of the Qt framework to split the layout from the implementation of the functionalities. It is possible to configure the settings to import a stylesheet file which will change the application layout.

To do it, you need to add to your settings file the variable PYFORMS_STYLESHEET with the path to the css file you want to use:

```
PYFORMS_STYLESHEET = 'style.css'
```

You may would like also to adapt the layout for a specific operating system.

The next variables will allow to do this. You can complement the style configured in PYFORMS_STYLESHEET with a stylesheet for a specific operating system.

```
PYFORMS_STYLESHEET_DARWIN = 'style_darwin.css'
PYFORMS_STYLESHEET_LINUX = 'style_linux.css'
PYFORMS_STYLESHEET_WINDOWS = 'style_window.css'
```

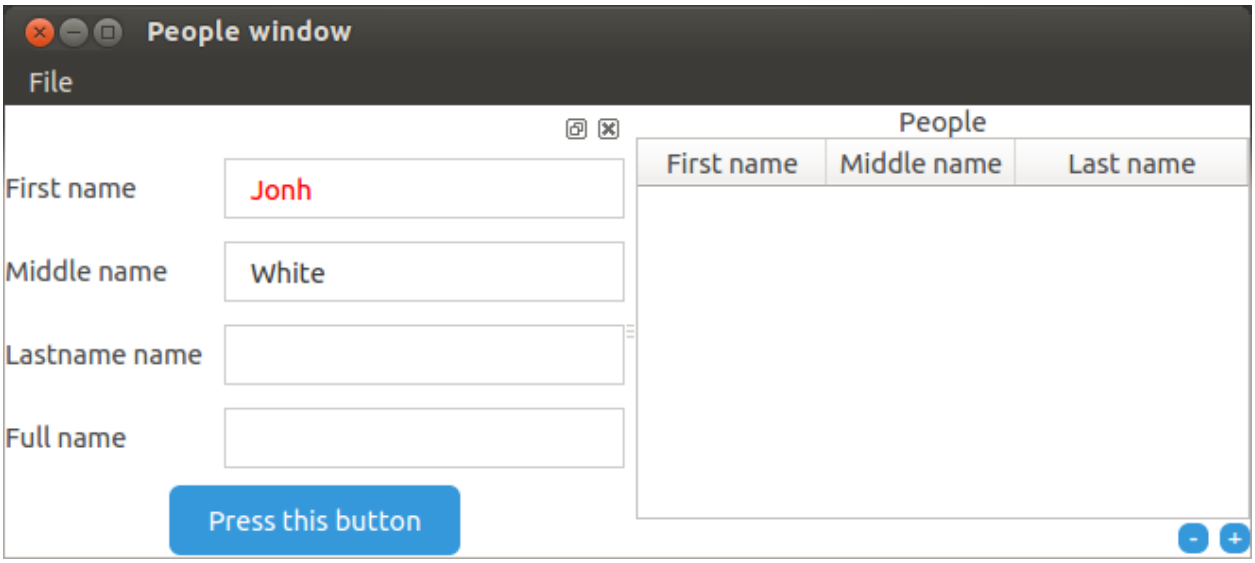
Check the example: style.css

```
QMainWindow{
        background-color: white;
}

QLabel{
        min-width: 110px;
}

QLineEdit{
        min-width: 200px;
        border: 1px solid #CCC;
        height: 30px;
        padding-left: 10px;
}

QPushButton{
        background: #3498db;
        color: #ffffff;
        padding: 10px 20px 10px 20px;

    border-radius: 6px;
}

QPushButton:hover {
```

(continues on next page)

(continued from previous page)

```
  background: #3cb0fd;
}

/*Use the # and the name of the variable to access to a specific the Control*/
#_firstnameField QLineEdit{
        color:red;
}
```



CHAPTER

EIGHT

PYTHON

8.1 BaseWidget

8.1.1 Overview

The BaseWidget class is the base class of all pyforms applications.

8.1.2 API

class pyforms_gui.basewidget.**BaseWidget**(**args*, ***kwargs*)

Bases: PyQt5.QtWidgets.QFrame

The class implements the most basic widget or window.

init_form()

Generate the module Form

generate_panel(*formset*)

Generate a panel for the module form with all the controls formset format example: [('_video', '_arenas', '_run'), {"Player":['_threshold', "_player", "=", "_results", "_query"], "Background image":[(' ', '_selectBackground', '_paintBackground'), '_image']}, "_progress"] tuple: will display the controls in the same horizontal line list: will display the controls in the same vertical line dict: will display the controls in a tab widget '||': will split the controls in a horizontal line '=': will split the controls in a vertical line @param formset: Form configuration @type formset: list

show(*self*)

close(*self*) → bool

input_text(*msg*, *title=''*, *default=None*)

input_double(*msg*, *title=''*, *default=0*, *min=-2147483647*, *max=2147483647*, *decimals=1*)

input_int(*msg*, *title=''*, *default=0*, *min=-2147483647*, *max=2147483647*)

question(*msg, title=None, buttons=['no', 'yes']*)

message(*msg*, *title=None*, *msg_type=None*)

success(*msg*, *title=None*)

info(*msg*, *title=None*)

warning(*msg*, *title=None*)

alert(*msg*, *title=None*)

critical(*msg*, *title=None*)

about(*msg*, *title=None*)

aboutQt(*msg*, *title=None*)

message_popup(*msg*, *title=''*, *buttons=None*, *handler=None*, *msg_type='success'*)

success_popup(*msg*, *title=''*, *buttons=None*, *handler=None*)

info_popup(*msg*, *title=''*, *buttons=None*, *handler=None*)

warning_popup(*msg*, *title=''*, *buttons=None*, *handler=None*)

alert_popup(*msg*, *title=''*, *buttons=None*, *handler=None*)

set_margin(*margin*)

property controls
Return all the form controls from the the module

property form_has_loaded

property parent_widget

property form

property title

property formset

property uid

closeEvent(*self*, *QCloseEvent*)

8.2 Controls

A form Control is a UI interface for the user to interact with the application.

Bellow we can find the description of all the Controls implemented in the PyForms library.

8.2.1 ControlBase

class pyforms_gui.controls.control_base.**ControlBase**(**args*, ***kwargs*)
Bases: object

All the Controls inherit from this Control, therefore you can find its functions and properties in all the other controls listed below.

Parameters

- **label** (*str*) – Control label. Default = ''.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.

- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

init_form()
Load the control UI and initiate all the events.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

show()
Show the control

hide()
Hide the control

add_popup_submenu(*label*, *submenu=None*)
It returns a new sub popup menu. If submenu is open the menu is added to the main popup menu.

Parameters

- **label** (*str*) – Label of the option
- **submenu** (*QMenu*) – Parent submenu to which the option should be added. If no value is set, then the option will be added to the main popup menu.

add_popup_menu_option(*label*, *function_action=None*, *key=None*, *icon=None*, *menu=None*)
Add an option to the Control popup menu.

Parameters

- **label** (*str*) – Label of the option
- **function_action** (*function*) – The function that should be executed when the menu is selected.
- **key** (*str*) – Short key.
- **or str icon** (*QIcon*) – Icon.
- **submenu** (*QMenu*) – Parent submenu to which the option should be added. If no value is set, then the option will be added to the main popup menu.

```
control.add_popup_menu_option('option 0', function_action=self._do_something)
submenu1 = control.add_popup_submenu('menu 1')
submenu2 = control.add_popup_submenu('menu 2', submenu=submenu1)
control.add_popup_menu_option('option 1', function_action=self._do_something,␣
↪key='Control+Q', submenu=submenu2)
```

changed_event()
Function called when ever the Control value is changed. The event function should return True if the data was saved with success.

about_to_show_contextmenu_event()
Function called before the Control popup menu is opened.

property enabled
Returns or set if the control is enable or disable.

property value
This property returns or set what the control should manage or store.

property name
This property returns or set the name of the control.

property label
Returns or sets the label of the control.

property parent
Returns or set the parent basewidget where the Control is.

property visible
Return the control visibility.

property help
Returns or set the tip box of the control.

property readonly
Set and return the control readonly state.

property form
Returns the QWidget of the control.

8.2.2 ControlBoundingSlider

class pyforms_gui.controls.control_boundingslider.**ControlBoundingSlider**(**args*, ***kwargs*)

Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **default** (*tupple*) – The default value is a list containing in the first element the lower value and in the second element the upper value. Default = [20,40].
- **horizontal** (*bool*) – Flag indicating if the Bounding slider should be draw horizontally or vertically. Default = True.
- **show_spinboxes** (*bool*) – Show or hide the spinboxes. Default = True
- **minimum** (*float*) – Defines the minimum value that can be selected.
- **maximum** (*float*) – Defines the maximum value that can be selected.
- **convert_2_int** (*bool*) – Flag to define if the control should return floats or integers.

property label
Returns or sets the label of the control.

property value
Sets and gets the control value. It should be a list or tuple of 2 values.

property min
Sets and gets the minimum value possible.

property max
Sets and gets the maximum value possible.

property scale
Sets and gets the scale value.

property convert_2_int
Flag to define if the control should return floats or integers.

8.2.3 ControlButton

class pyforms_gui.controls.control_button.**ControlButton**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

...

Parameters

- **icon** (*str*) – Button icon
- **checkable** (*bool*) – Flag to set the button checkable.

click()
Trigger a click event

property icon
Sets and gets the icon of the button.

property value
Sets and gets the value of the Button. The value should be a function

property checked
Sets and gets the button checked state

8.2.4 ControlCheckBox

class pyforms_gui.controls.control_checkbox.**ControlCheckBox**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ''.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.

- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

property value
This property returns or set what the control should manage or store.

8.2.5 ControlCheckBoxList

class pyforms_gui.controls.control_checkboxlist.**ControlCheckBoxList**(**args*, ***kwargs*)

Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ''.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

save_form(*data={}*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

item_changed(*item*)

clear()

refresh()

selection_changed_event()

property count

property checked_indexes

property value
This property returns or set what the control should manage or store.

property selected_row_index

property items

8.2.6 ControlCodeEditor

class pyforms_gui.controls.control_codeeditor.**ControlCodeEditor**(**args*, ***kwargs*)

Bases: *pyforms_gui.controls.control_base.ControlBase*

Control that offers a code editor with pretty-print and line numbers and a save button

Parameters

- **label** –
- **default** –
- **helptext** –

ARROW_MARKER_NUM = 8

on_margin_clicked(*nmargin*, *nline*, *modifiers*)
On margin clicked, toggle marker for the line the margin was clicked on :param nmargin: :type nmargin: :param nline: :type nline: :param modifiers: :type modifiers:

on_modification_changed()
On modification change, re-enable save button

on_save_changes()
On button save clicked, save changes made on the code editor to file

on_discard_changes()

discard_event()

key_pressed_event(*event*)
Override KeyPressed event as you like :param event: key event

property is_modified

property lexer

property value
This property returns or set what the control should manage or store.

property changed_event
Function called when ever the Control value is changed. The event function should return True if the data was saved with success.

8.2.7 ControlCombo

class pyforms_gui.controls.control_combo.**ControlCombo**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QWidget

This class represents a wrapper to the combo box

clear()

add_item(*label*, *value=<class 'pyforms_gui.controls.control_combo.ValueNotSet'>*)

get_item_index_by_name(*item_name*)
Returns the index of the item containing the given name :param item_name: item name in combo box :type item_name: string

count()

show()
Show the control

hide()
Hide the control

current_index_changed_event(*index*)
Called when the user chooses an item in the combobox and the selected choice is different from the last one selected. @index: item's index

activated_event(*index*)
Called when the user chooses an item in the combobox. Note that this signal happens even when the choice is not changed @index: item's index

highlighted_event(*index*)

edittext_changed_event(*text*)

property form
Returns the QWidget of the control.

property current_index

property values

property items

property value
This property returns or set what the control should manage or store.

property text

property label
Returns or sets the label of the control.

8.2.8 ControlDir

class pyforms_gui.controls.control_dir.**ControlDir**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

click()

finishEditing()
Function called when the lineEdit widget is edited

property value
This property returns or set what the control should manage or store.

property label
Returns or sets the label of the control.

8.2.9 ControlDockWidget

class pyforms_gui.controls.control_dockwidget.**ControlDockWidget**(**args*, ***kwargs*)

Bases: *pyforms_gui.controls.control_emptywidget.ControlEmptyWidget*

SIDE_LEFT = 'left'

SIDE_RIGHT = 'right'

SIDE_TOP = 'top'

SIDE_BOTTOM = 'bottom'

SIDE_DETACHED = 'detached'

property label
Returns or sets the label of the control.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

load_form(*data*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

show()
Show the control

hide()
Hide the control

8.2.10 ControlEmptyWidget

class pyforms_gui.controls.control_emptywidget.**ControlEmptyWidget**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QWidget

property value
This property returns or set what the control should manage or store.

property form
Returns the QWidget of the control.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

show()
Show the control

hide()
Hide the control

8.2.11 ControlFile

class pyforms_gui.controls.control_file.**ControlFile**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

finishEditing()
Function called when the lineEdit widget is edited

click()

property value
This property returns or set what the control should manage or store.

property label
Returns or sets the label of the control.

8.2.12 ControlFilesTree

class pyforms_gui.controls.control_filestree.**ControlFilesTree**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

property value
This property returns or set what the control should manage or store.

8.2.13 ControlImage

class pyforms_gui.controls.control_image.**ControlImage**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

property value
This property returns or set what the control should manage or store.

property double_click_event

property click_event

property drag_event

property end_drag_event

property key_release_event

8.2.14 ControlLabel

class pyforms_gui.controls.control_label.**ControlLabel**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

property selectable

property form
Returns the QWidget of the control.

property value
This property returns or set what the control should manage or store.

8.2.15 ControlList

class pyforms_gui.controls.control_list.**ControlList**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QWidget

This class represents a wrapper to the table widget It allows to implement a list view

CELL_VALUE_BEFORE_CHANGE = None

clear(*headers=False*)

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

set_value(*column*, *row*, *value*)

get_value(*column*, *row*)

resize_rows_contents()

get_currentrow_value()

get_cell(*column*, *row*)

set_sorting_enabled(*value*)
Enable or disable columns sorting

Parameters **value** (*bool*) – True to enable sorting, False otherwise

data_changed_event(*row*, *col*, *item*)

item_selection_changed_event()

current_cell_changed_event(*next_row*, *next_col*, *previous_row*, *previous_col*)

current_item_changed_event(*current*, *previous*)

cell_double_clicked_event(*row*, *column*)

property height

property horizontal_headers

property word_wrap

property readonly
Set and return the control readonly state.

property select_entire_row

property rows_count

property columns_count

property value
This property returns or set what the control should manage or store.

property selected_rows_indexes

property selected_row_index

property label
Returns or sets the label of the control.

property form
Returns the QWidget of the control.

property icon_size

property autoscroll

property resizecolumns

tableWidgetCellChanged(*nextRow*, *nextCol*, *previousRow*, *previousCol*)

tableWidgetItemChanged(*current*, *previous*)

tableWidgetItemSelectionChanged()

tableWidgetCellDoubleClicked(*row*, *column*)
(From PyQt) This signal is emitted whenever a cell in the table is double clicked. The row and column specified is the cell that was double clicked.

Besides firing this signal, we save the current value, in case the user needs to know the old value. :param row: :param column: :return:

empty_signal(**args*, ***kwargs*)
Use this function if you want to disconnect a signal temporarily

---

8.2.16 ControlPlayer

8.2.17 ControlMatplotlib

class pyforms_gui.controls.control_matplotlib.**ControlMatplotlib**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QWidget

property value
This property returns or set what the control should manage or store.

draw()

on_draw(*figure*)
Redraws the figure

property fig

property form
Returns the QWidget of the control.

8.2.18 ControlMdiArea

class pyforms_gui.controls.control_mdiarea.**ControlMdiArea**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QMdiArea

The ControlMdiArea wraps a QMdiArea widget which provides an area in which MDI windows are displayed.

property show_subwin_close_button

property label
Returns or sets the label of the control.

property form
Returns the QWidget of the control.

8.2.19 ControlNumber

class pyforms_gui.controls.control_number.**ControlNumber**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **minimum** (*int*) – Minimum value.
- **maximum** (*int*) – Maximum value.
- **default** (*float*) – Set the value. Default = 0.
- **decimals** (*int*) – Decimals precision.
- **step** (*float*) – Step jump value.

update_event(*value*)

property label
Returns or sets the label of the control.

property value
This property returns or set what the control should manage or store.

property min

property max

property decimals

property step

8.2.20 ControlPassword

class pyforms_gui.controls.control_password.**ControlPassword**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_text.ControlText*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.

- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

8.2.21 ControlOpenGL

class pyforms_gui.controls.control_opengl.**ControlOpenGL**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ''.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

repaint()

reset_zoom_and_rotation()

property value
This property returns or set what the control should manage or store.

property clear_color

property width

property height

8.2.22 ControlProgress

class pyforms_gui.controls.control_progress.**ControlProgress**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

property label
Returns or sets the label of the control.

property value
This property returns or set what the control should manage or store.

property min

property max

8.2.23 ControlSlider

class pyforms_gui.controls.control_slider.**ControlSlider**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

valueChanged(*value*)

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

property value
This property returns or set what the control should manage or store.

property min

property max

8.2.24 ControlText

class pyforms_gui.controls.control_text.**ControlText**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

finishEditing()
Function called when the lineEdit widget is edited

key_pressed_event(*evt*)

property value
This property returns or set what the control should manage or store.

property label
Returns or sets the label of the control.

property readonly
Set and return the control readonly state.

8.2.25 ControlTextArea

class pyforms_gui.controls.control_textarea.**ControlTextArea**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

finishEditing()
Function called when the lineEdit widget is edited

property value
This property returns or set what the control should manage or store.

property readonly
Set and return the control readonly state.

property autoscroll

8.2.26 ControlToolBox

class pyforms_gui.controls.control_toolbox.**ControlToolBox**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

property value
This property returns or set what the control should manage or store.

set_item_enabled(*index*, *enabled*)
Enable or disable an item

is_item_enabled(*index*)
Check if an item is enabled or disabled

8.2.27 ControlToolButton

class pyforms_gui.controls.control_toolbutton.**ControlToolButton**(**args*, ***kwargs*)

Bases: *pyforms_gui.controls.control_base.ControlBase*

click()

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

property label
Returns or sets the label of the control.

property icon

property value
This property returns or set what the control should manage or store.

property checked

8.2.28 ControlTree

class pyforms_gui.controls.control_tree.**ControlTree**(**args*, ***kwargs*)

Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QTreeWidget

This class represents a wrapper to the QTreeWidget

save_form(*data*, *path=None*)
Save a value of the control to a dictionary.

Parameters

- **data** (*dict*) – Dictionary where the control value should be saved.
- **path** (*str*) – Optional parameter that can be used to load the data.

load_form(*data*, *path=None*)
Loads the value of the control.

Parameters

- **data** (*dict*) – It is a dictionary with the required information to load the control.
- **path** (*str*) – Optional parameter that can be used to save the data.

add_popup_menu_option(*label=''*, *function_action=None*, *key=None*, *item=None*, *icon=None*, *submenu=None*)
Add an option to the Control popup menu @param label: label of the option. @param function_action: function called when the option is selected. @param key: shortcut key @param key: shortcut key

clear(*self*)

expand_item(*item*, *expand=True*, *parents=True*)

create_child(*name*, *parent=None*, *icon=None*)
Create a new child for to the parent item. If the parent is None it add to the root.

item_changed_event(*item*)

item_selection_changed_event()

item_double_clicked_event(*item*)

key_press_event(*event*)

rows_inserted_event(*parent*, *start*, *end*)
This event is called every time a new row is added to the tree

property show_header

property selected_rows_indexes

property selected_row_index

property selected_item

property form
Returns the QWidget of the control.

property value
This property returns or set what the control should manage or store.

property icon_size

rowsInserted(*self*, *QModelIndex*, *int*, *int*)

selectionChanged(*self*, *QItemSelection*, *QItemSelection*)

keyPressEvent(*self*, *QKeyEvent*)

about_to_show_contextmenu_event()
Function called before open the Control popup menu

clone_item(*parent*, *item*, *copy_function=None*)

clone_tree(*tree*, *copy_function=None*)

8.2.29 ControlTreeView

class pyforms_gui.controls.control_treeview.**ControlTreeView**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QTreeView

default_width = None

item_selection_changed_event(*selected*, *deselected*)

item_double_clicked_event(*evt*)

property selected_row_index

property selected_item

property value
This property returns or set what the control should manage or store.

property form
Returns the QWidget of the control.

8.2.30 ControlVisVis

class pyforms_gui.controls.control_visvis.**ControlVisVis**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.
- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

refresh()

paint(*visvis*)

property legend

property show_grid

property title

property xlabel

property ylabel

property zlabel

property value
This property returns or set what the control should manage or store.

8.2.31 ControlVisVisVolume

class pyforms_gui.controls.control_visvisvolume.**ControlVisVisVolume**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*

Parameters

- **label** (*str*) – Control label. Default = ‘’.

- **helptext** (*str*) – Text shown when the mouse is over the control. Default = None.
- **default** (*str*) – Initial value of the control. Default = None.
- **visible** (*bool*) – Flag to set the control visible or hidden. Default = True.
- **enabled** (*bool*) – Flag to set the control enabled or Disabled. Default = True.
- **readonly** (*bool*) – Flag to set the control readonly. Default = False.
- **changed_event** (*function*) – Function to call whenever the control value is updated. Default = None.

property color_map

refresh()

property value
This property returns or set what the control should manage or store.

property colors_limits

property visvis

8.2.32 ControlWeb

class pyforms_gui.controls.control_web.**ControlWeb**(**args*, ***kwargs*)
Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWebEngineWidgets.QWebEngineView

load_finished_event(*ok*)

property value
This property returns or set what the control should manage or store.

property html

property form
Returns the QWidget of the control.

8.2.33 ControlEventTimeline

class pyforms_gui.controls.control_event_timeline.control_eventtimeline.**ControlEventTimeli**

Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QWidget

Timeline events editor

Short keys:

- **Control + Left**: Move event to the left.
- **Control + Right**: Move event to the right.
- **Delete**: Delete an event.
- **L**: Lock an event.

- **Control + Up**: Move an event up.
- **Control + Down**: Move an event down.
- **Shift + Control + Left**: Move an event end time to the left.
- **Shift + Control + Right**: Move an event end to the right.
- **Shift + Left**: Move an event beginning to the left.
- **Shift + Right**: Move an event beginning to the right.
- **S**: First press, mark the beginning of an event, Second press, create an event ending in the current cursor time.
- **A**: Move the cursor to the left.
- **D**: Move the cursor to the right.
- **Q**: Select the previous event in the selected row.
- **E**: Select the next event in the selected row.

move_selected_event_or_pointer_left()
Move the selected event or pointer to the left. :return:

move_selected_event_or_pointer_right()
Move the selected event or pointer to the right. :return:

remove_selected_event()
Remove the selected event. :return:

toggle_selected_event_lock()
Toggle the lock of the selected event. :return:

select_next_event()
Select the next event. :return:

select_previous_event()
Select the previous event. :return:

select_first_event()
Select the first event. :return:

select_last_event()
Select the last event. :return:

move_selected_event_up()
Move the selected event to the track above. :return:

move_selected_event_down()
Move the selected event to the track bellow. :return:

move_selected_event_end_left()
Move the selected event end to the left. :return:

move_selected_event_end_right()
Move the selected event end to the right. :return:

move_selected_event_begin_left()
Move the selected event begin to the left. :return:

move_selected_event_begin_right()
Move the selected event begin to the right. :return:

open_and_close_event()
Open and close and event. The first time the function is called an event is opened. On the second call the event is closed and added to the timeline. :return:

add_event(*begin*, *end*, *title=''*, *row=0*, *track=None*)

Parameters

- **begin** – Initial frame
- **end** – Last frame
- **title** – Event title
- **row** – Row to which the event should be added.

add_graph(*name*, *data*)

Parameters

- **name** –
- **data** –

Returns

add_track(*title=''*, *color=None*)
Add a new track. :param str title: Title of the track. :param QColor color: Default color of the events in the track. :return: Return the added track.

get_track(*title*)
Get a track by its title :param str title: Title of the track. :return: Return the track with the matching title.

rename_graph(*graph_index*, *newname*)
Rename a graph by index. :param int graph_index: Index of the graph to rename. :param str newname: New name

import_graph_csv(*filepath*, *separator=';'*, *ignore_rows=0*)
Import a new graph from a csv file. :param filename: :param separator: :param ignore_rows: :return:

export_csv_file(*filename*)

import_csv_file(*filename*)

import_csv(*csvfile*)

Parameters **csvfile** –

open_graphs_properties()
Opens the graphs properties.

open_import_graph_win(*filepath*, *frame_col=0*, *val_col=1*)
Open a window to import a graph from a csv file. :param str filepath: Path of the file to import. :param int frame_col: Column corresponding to the frames number in the csv file. :param int val_col: Column corresponding to the values in the csv file.

mouse_moveover_timeline_event(*event*)

property pointer_changed_event

property timeline_widget

property value
This property returns or set what the control should manage or store.

property max

property form
Returns the QWidget of the control.

property rows

property graphs

property key_release_event

about_to_show_contextmenu_event()
Hide and show context menu options.

clean()

8.2.34 ControlEventsGraph

class pyforms_gui.controls.control_events_graph.control_eventsgraph.**ControlEventsGraph**(*label*, *default=*, *min=*, *max=*, ***kw*

Bases: *pyforms_gui.controls.control_base.ControlBase*, PyQt5.QtWidgets.QWidget

Timeline events editor

Parameters

- **label** –
- **default** –
- **min** –
- **max** –
- **kwargs** –

add_track(*title=None*)

Parameters **title** –

add_event(*begin*, *end*, *title=''*, *track=0*, *color='#FFFF00'*)

Parameters

- **begin** –
- **end** –
- **title** –
- **track** –
- **color** –

Returns

get_export_filename()

export_csv(*filename*)
Export annotations to a file. :param str filename: filename to open

repaint()

property changed_event
Function called when ever the Control value is changed. The event function should return True if the data was saved with success.

property value
This property returns or set what the control should manage or store.

property form
Returns the QWidget of the control.

property tracks

property tracks_height

property scale

8.3 Settings

Pyforms is using the confapp library to manage it settings. Here it is described some of the settings of the library.

8.3.1 General configurations

PYFORMS_MODE = os.environ.get('PYFORMS_MODE', 'GUI')

It defines the mode that the pyforms should run. Currently pyforms can run as **GUI** or **TERMINAL** mode.

PYFORMS_LOG_HANDLER_FILE_LEVEL = logging.DEBUG

Logging level.

PYFORMS_LOG_HANDLER_CONSOLE_LEVEL = logging.INFO

Logging level.

8.3.2 GUI layout

PYFORMS_STYLESHEET = None

Path to the stylesheet file of the application.

PYFORMS_STYLESHEET_DARWIN = None

PYFORMS_STYLESHEET_LINUX = None

PYFORMS_STYLESHEET_WINDOWS = None

Frequently it is necessary to adapt the layout of an application for each operating system. These variables allow you to do just that. For each operating system you can define a stylesheet that will complement the default stylesheet for a specific OS.

8.3.3 Controls

PYFORMS_CONTROL_CODE_EDITOR_DEFAULT_FONT_SIZE = '12'

PYFORMS_CONTROL_EVENTS_GRAPH_DEFAULT_SCALE = 1

PYFORMS_CONTROLPLAYER_FONT = 9

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z