Pyforms GUI Documentation

Release 4.9.2

Ricardo Jorge Vieira Ribeiro

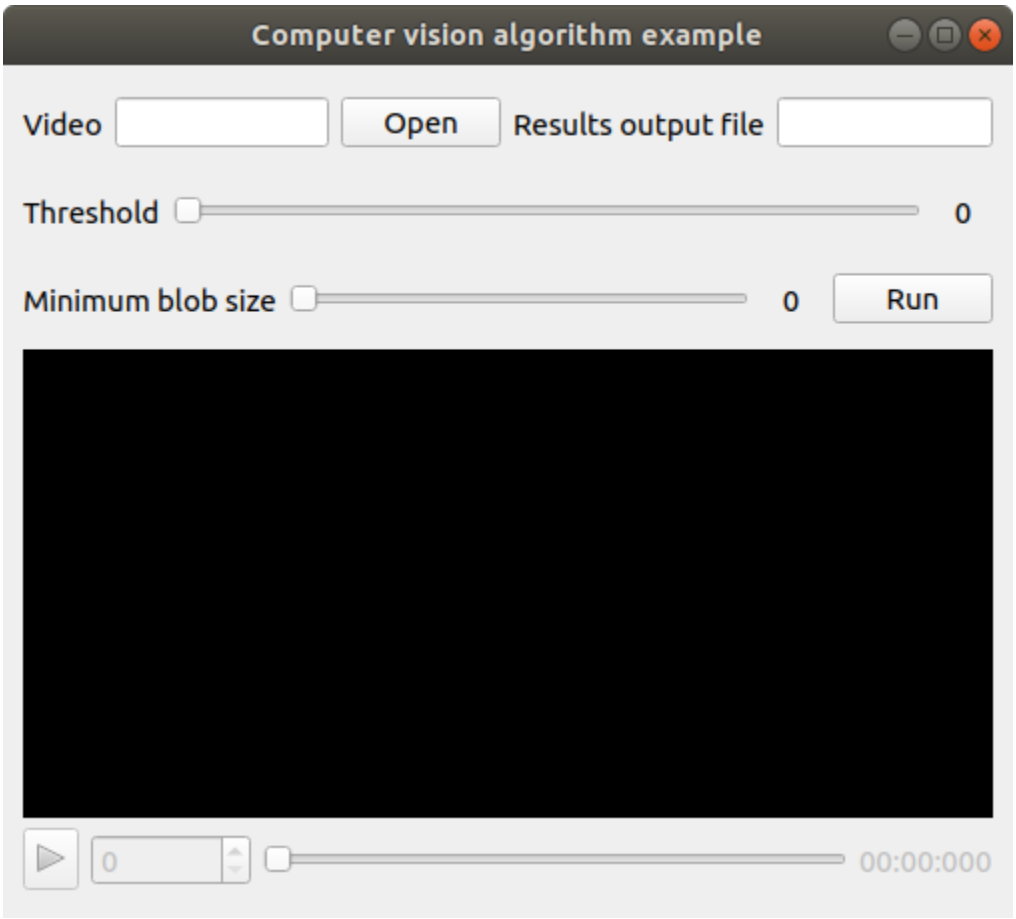
Jun 03, 2021

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
```

(continues on next page)

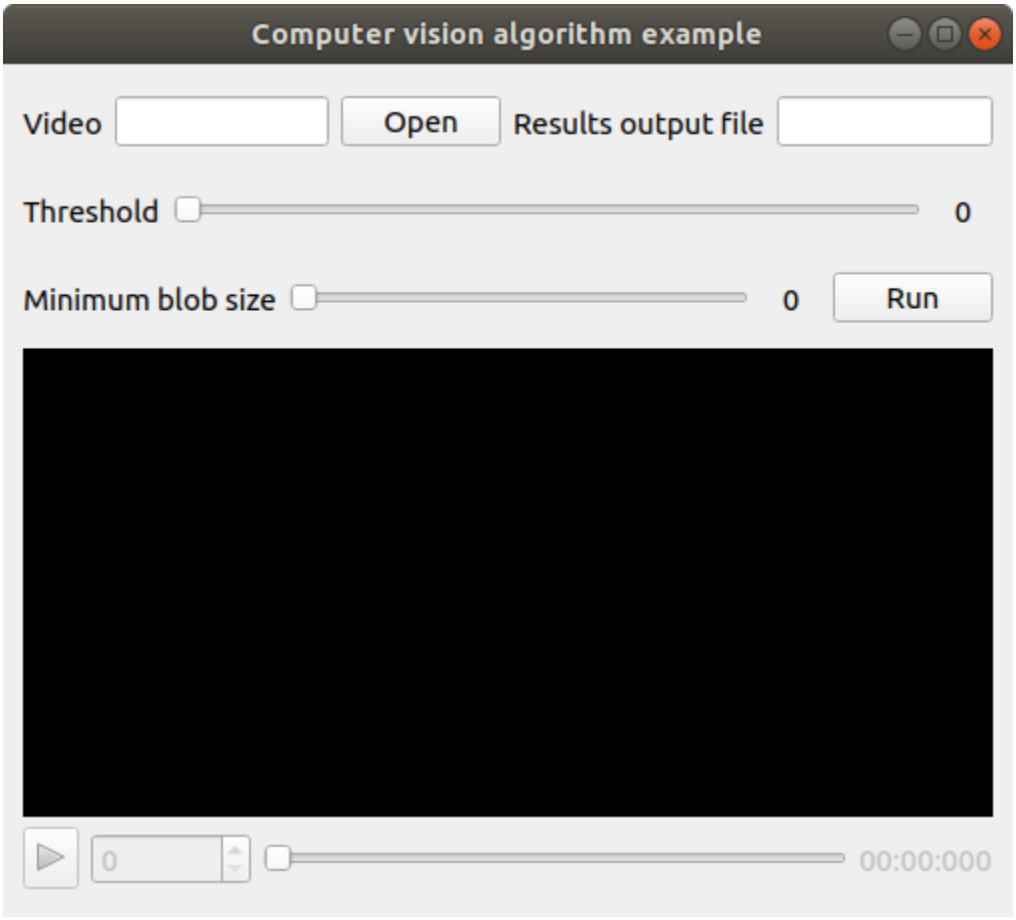
(continued from previous page)

```
        #Define the event called before showing the image in the player
        self._player.process_frame_event    = self.__process_frame

        #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
```

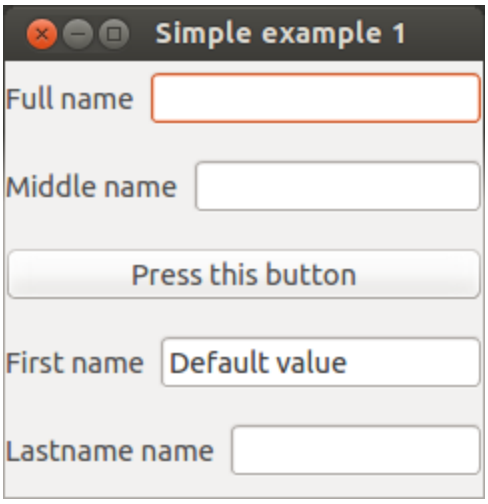
4.1.2 Create your application class

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:

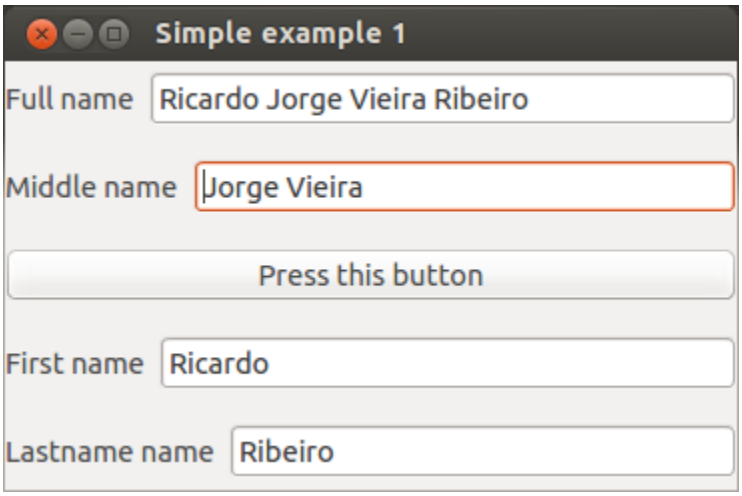
```
import pyforms
from   pyforms.basewidget import BaseWidget
from   pyforms.controls import ControlText
from   pyforms.controls import ControlButton

class SimpleExample1(BaseWidget):

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

        #Definition of the forms fields
        self._firstname  = ControlText('First name', 'Default value')
        self._middlename = ControlText('Middle name')
        self._lastname   = ControlText('Lastname name')
```

(continues on next page)

(continued from previous page)

```
        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

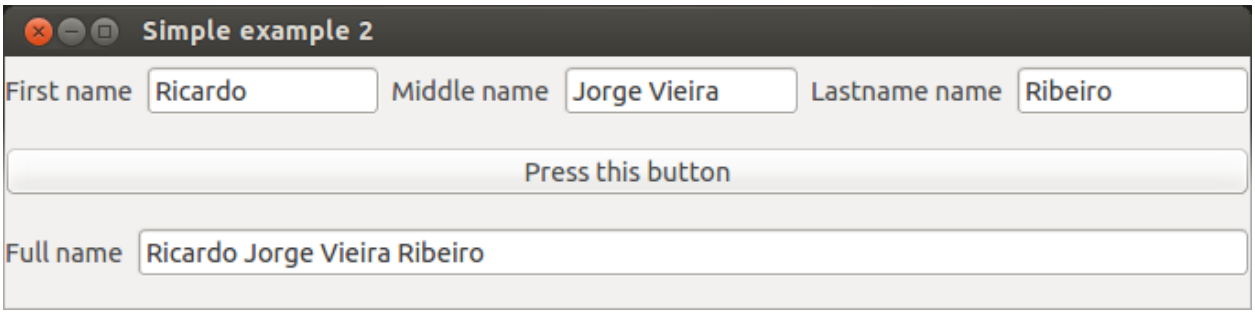
```
...

class SimpleExample1(BaseWidget):

    def __init__(self):
        ...

        #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
```

(continues on next page)

(continued from previous page)

```
    ...
```

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):
    ...

    def __openEvent(self):
        ...

    def __saveEvent(self):
        ...
```

(continues on next page)

(continued from previous page)

```
    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)

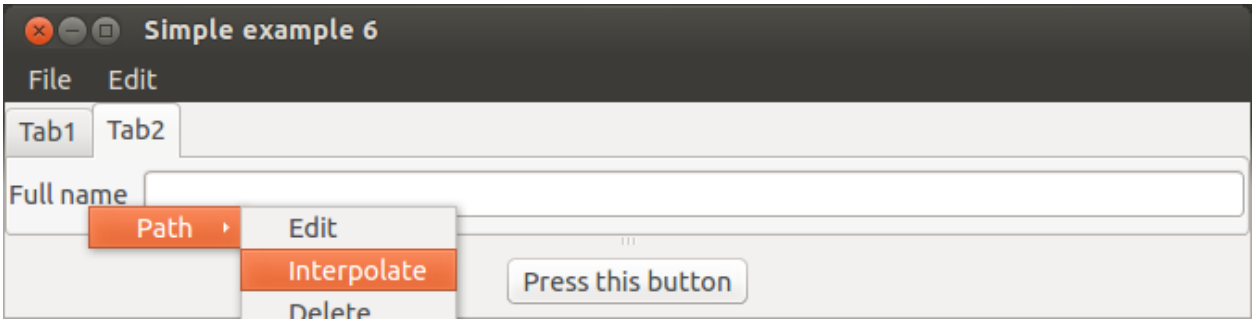
```
...

class SimpleExample1(BaseWidget):

    def __init__(self):
        ...

        self._fullname.addPopupSubMenuOption('Path',
            {
```

(continues on next page)

(continued from previous page)

```
                'Delete':           self.__dummyEvent,
                'Edit':             self.__dummyEvent,
                'Interpolate':      self.__dummyEvent
            })
    ...
```

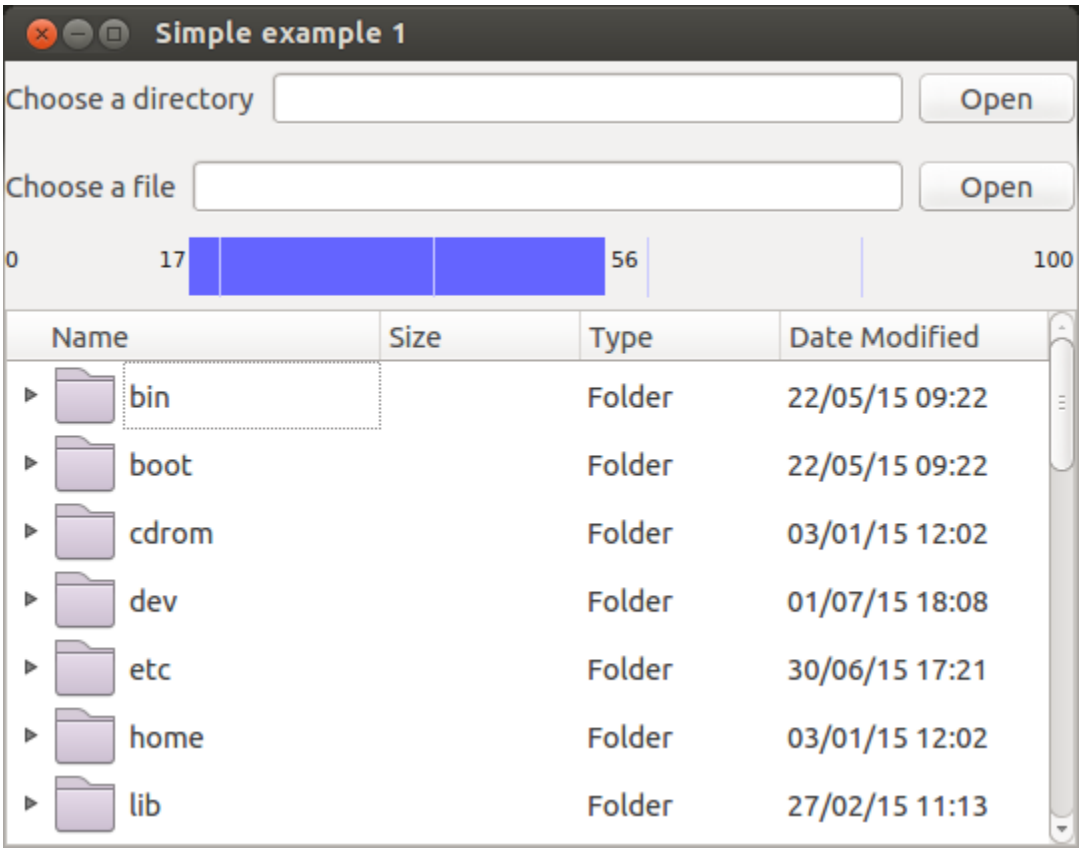
Result:

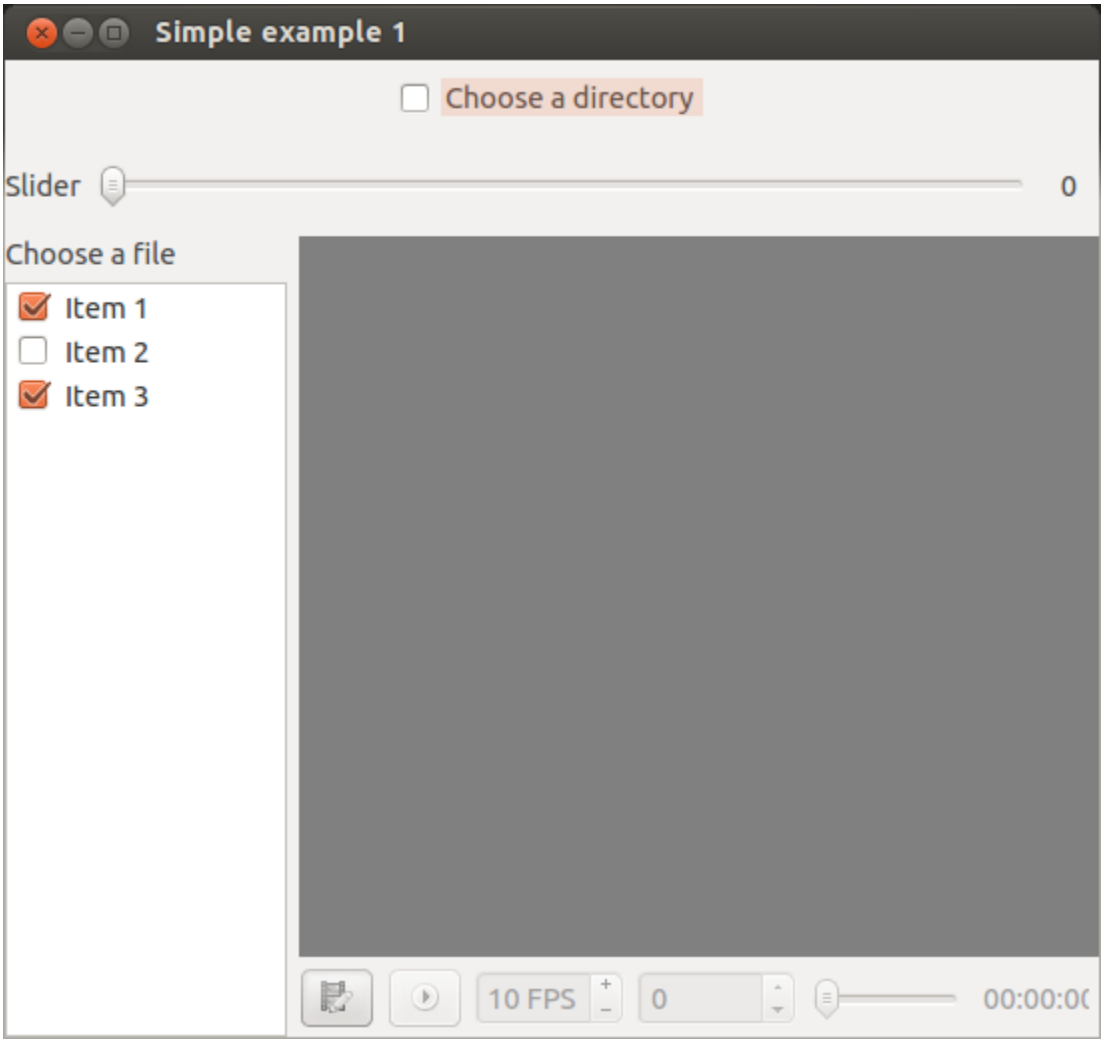


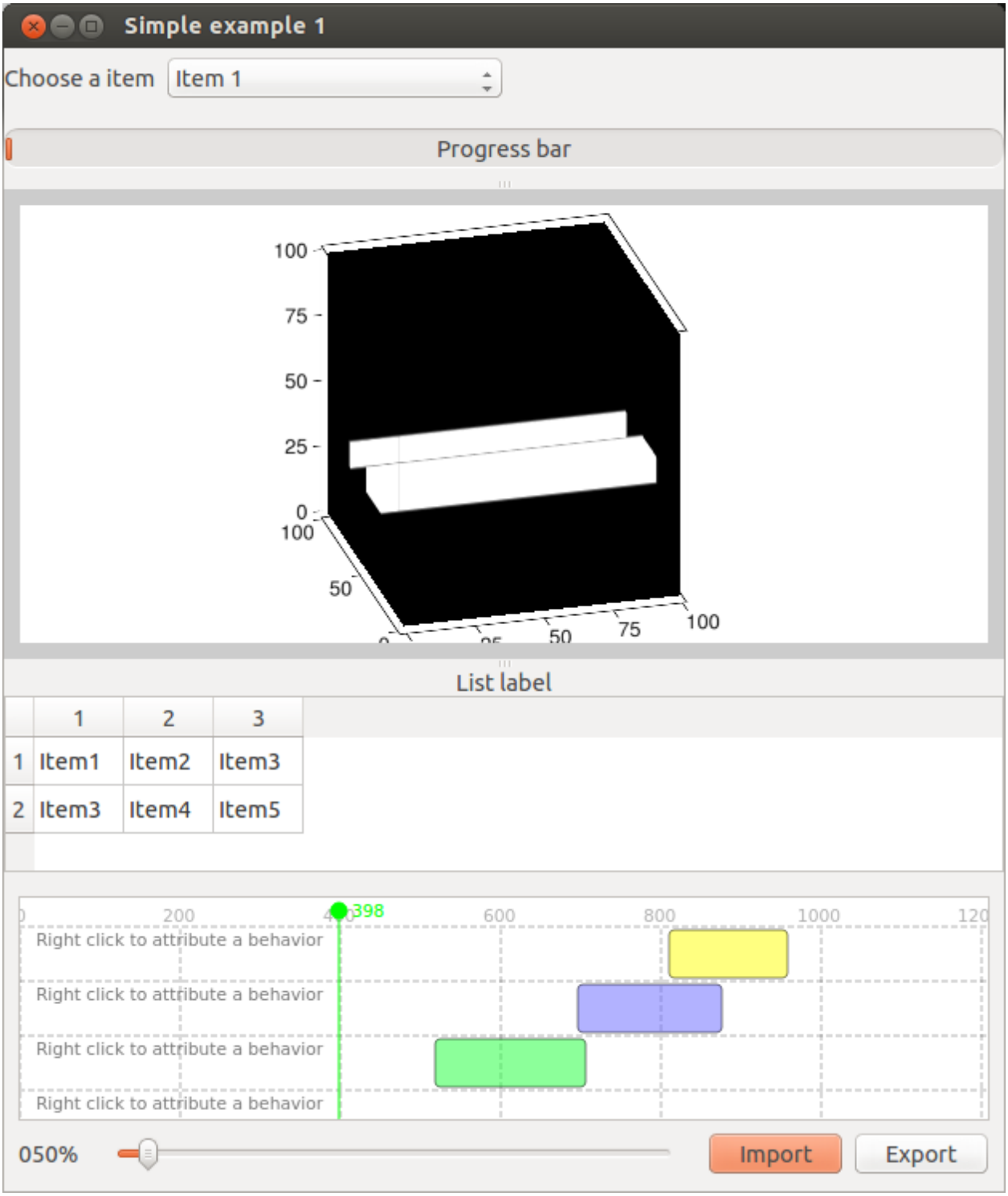
4.6 What's next?

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):
```

(continues on next page)

(continued from previous page)

```
        return self._people.pop(index)

    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)
```

(continues on next page)

(continued from previous page)

```
    #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.

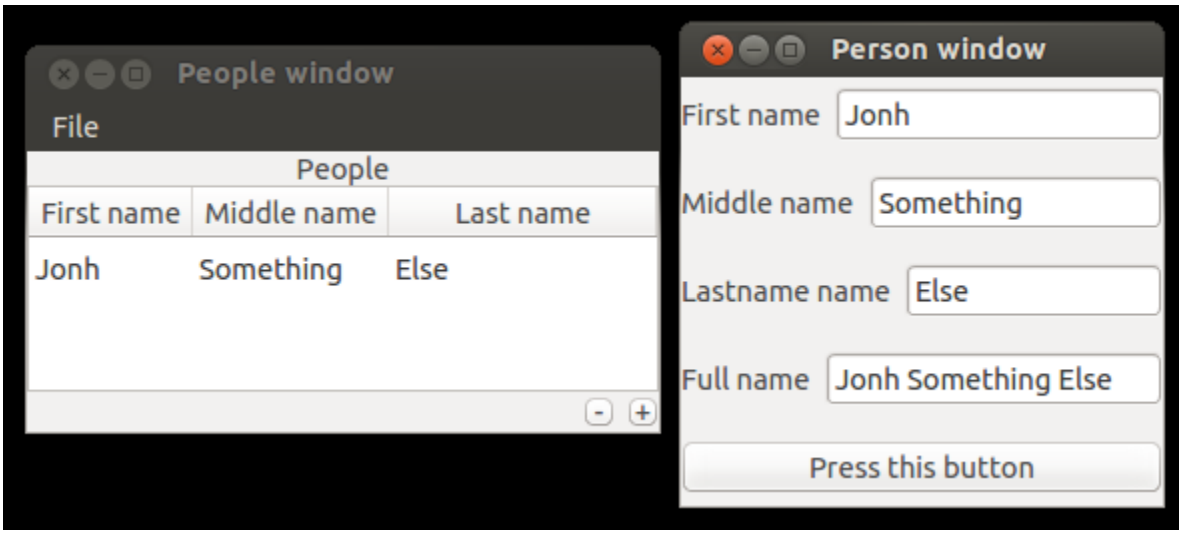
```
import pyforms
from pyforms.basewidget     import BaseWidget
from pyforms.controls       import ControlList
from People                 import People
from PersonWindow           import PersonWindow
from AddMenuFuntionality    import AddMenuFuntionality

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):
```

(continues on next page)

(continued from previous page)

```
        """
        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
        """
        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):
        """
```

(continues on next page)

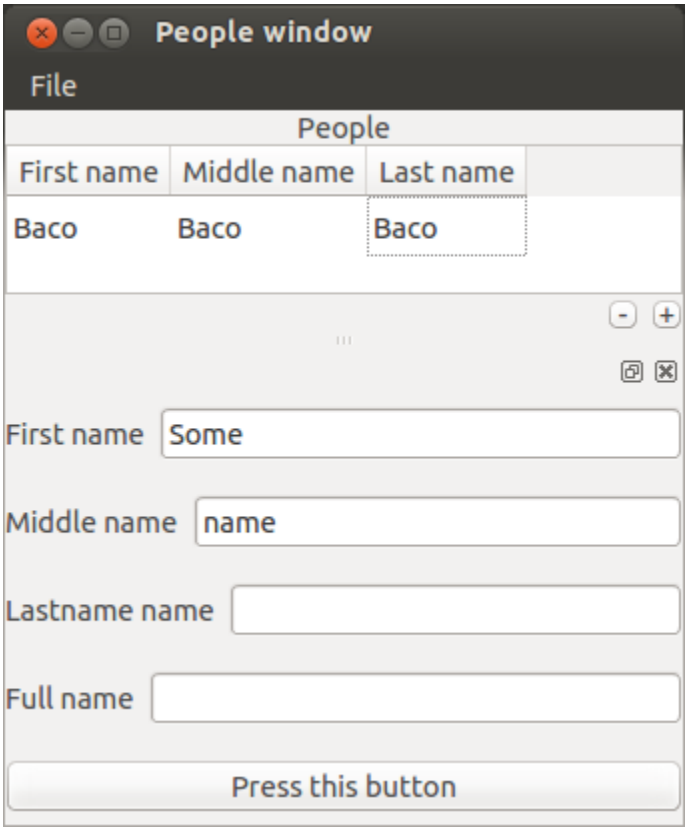
(continued from previous page)

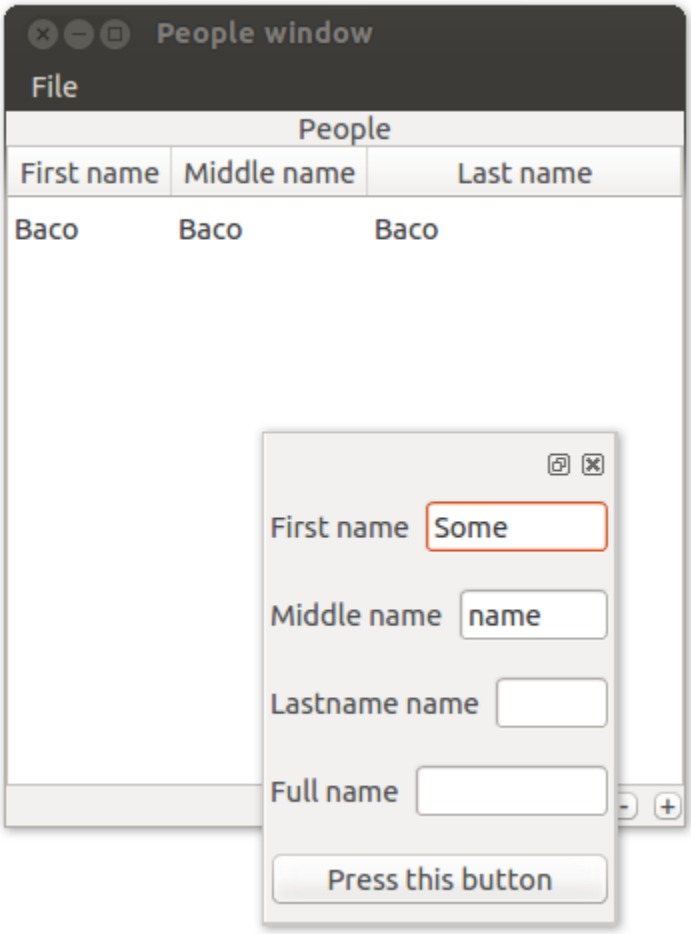
```
        Add person button event.
        """
        # A new instance of the PersonWindow is opened and shown to the user.
        win = PersonWindow()
        win.parent = self
        self._panel.value = win

...
```

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'
```

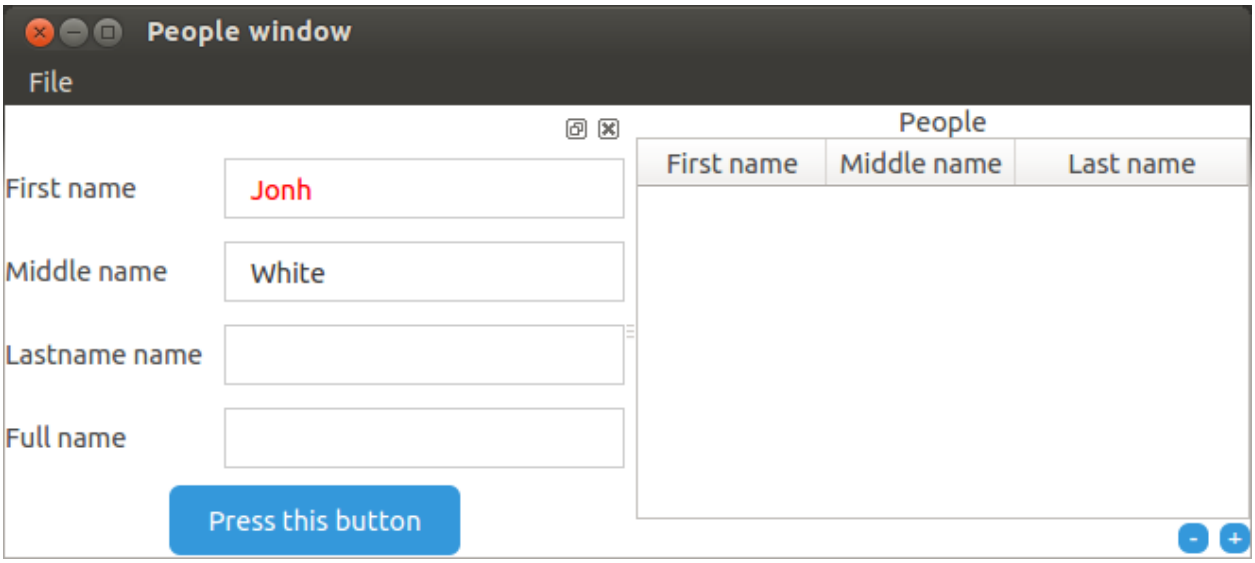
Check the example: style.css

```
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;
```

(continues on next page)

(continued from previous page)

```
}

QPushButton:hover {
  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

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

8.2.2 ControlBoundingSlider

8.2.3 ControlButton

8.2.4 ControlCheckBox

8.2.5 ControlCheckBoxList

8.2.6 ControlCodeEditor

8.2.7 ControlCombo

8.2.8 ControlDir

8.2.9 ControlDockWidget

8.2.10 ControlEmptyWidget

8.2.11 ControlFile

8.2.12 ControlFilesTree

8.2.13 ControlImage

8.2.14 ControlLabel

8.2.15 ControlList

8.2.16 ControlPlayer

8.2.17 ControlMatplotlib

8.2.18 ControlMdiArea

8.2.19 ControlNumber

8.2.20 ControlPassword

8.2.21 ControlOpenGL

8.2.22 ControlProgress

8.2.23 ControlSlider

8.2.24 ControlText

8.2.25 ControlTextArea

8.2.26 ControlToolBox

8.2.27 ControlToolButton

8.2.28 ControlTree

8.2.29 ControlTreeView

8.2.30 ControlVisVis

8.2.31 ControlVisVisVolume

8.2.32 ControlWeb

8.2.33 ControlEventTimeline

8.2.34 ControlEventsGraph

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

M

P