Stani's Python Editor

Python IDE with Blender, Kiki, PyChecker, wxGlade & XRC support

User Manual

22-08-2005

Table of Contents

1 Introduction

1.1 About

Stani's Python Editor

SPE is a cross-platform python IDE with auto indentation, auto completion, call tips, syntax coloring, syntax highlighting, uml viewer, class explorer, source index, automatic todo list, sticky notes, integrated pycrust shell, python file browser, recent file browser, drag&drop, context help, ... Special is its Blender support with a Blender 3d object browser and its ability to run interactively inside Blender.

SPE runs on Windows, Linux and Mac OS X.

SPE is extensible with wxGlade.

1.2 Plugins

SPE ships with

- wxGlade (gui designer)
- PyChecker (source code doctor)
- Kiki (regular expression console)

SPE also integrates with

- XRCed (gui designer)

1.3 Internet links

- Homepage: http://SPE.pycs.net
- Website: http://projects.blender.org/projects/SPE
- Screenshots: http://SPE.pycs.net/pictures/index.html
- Forum: http://projects.blender.org/forum/?groupid=30
- RSS feed: http://SPE.pycs.net/weblog/rss.xml

1.4 Copyright

1.5 License

SPE is released under the GPL. If you need SPE under another license, contact the author.

2 Installation

If you encounter any problems during or after installation, be sure to also read the FAQ.

2.1 Requirements

- Python 2.3+
  We recommend ActivePython distribution because of its excellent help files:
  http://www.activestate.com/Products/ActivePython/index.html
- wxPython 2.6+
  SPE follows the wxPython releases. Always use the latest wxPython release.

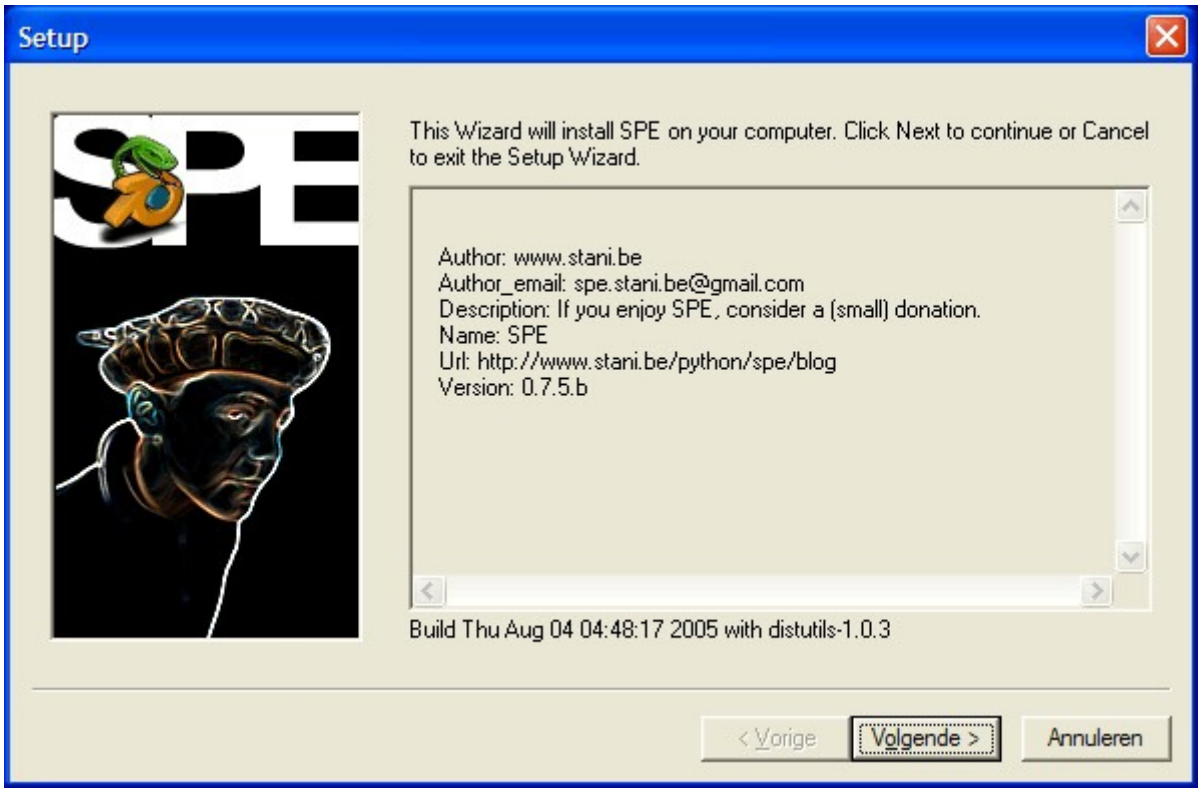
➢ Optional:

- Blender 2.37
  Cross-platform 3D software solution from modeling, animation, rendering and post-production to interactive creation and playback.
- Win32 extensions (Windows only)
  This module is needed to create shortcuts on Desktop and Start Menu during installation. This module is standard included in ActivePython

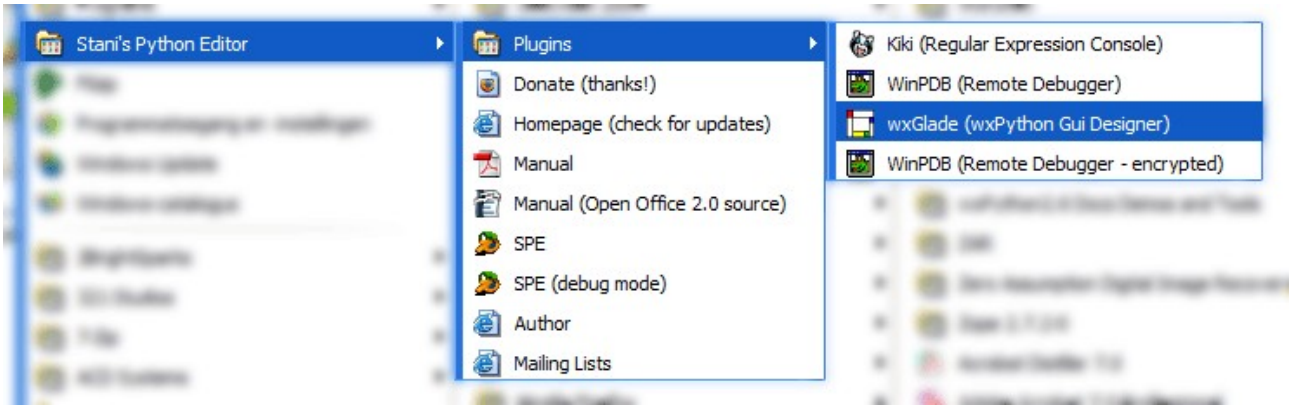
2.2 Windows

Install

Use the win32 installer (SPE-*-wx*-bl*.exe). Do not use the script spe_wininst.py, it will be called by the win32 installer automatically.



This will create icons on desktop & start menu and register SPE in the context menu of Windows Explorer:



Uninstall

Uninstall SPE with Add/Remove Progams in the Windows control panel. SPE should be mentioned as 'Python x.x SPE-x.x.x.x' If you don't find this entry, than do the action (Unix*) below.

2.3 Unix: Linux, FreeBSD, ...*

Install

Run the 'setup.py' script:

```
>python setup.py install
```

If you have any problems with permissions:

```
>sudo python setup.py install
```

This will install SPE in the standard library directory of python:
/usr/local/lib/pythonX.X/site-packages

A wrapper script called 'SPE' will be installed to PREFIX/bin. If necessary add PREFIX/bin to your PATH environment variable. PREFIX is determined by the install location of the modules, i.e. for the above PREFIX=/usr/local.

When SPE is launched in Blender, what might be missing in the PYTHONPATH, is /usr/local/lib/python2.3/site-packages. If you add this one in your .bashrc/.tcshrc/... to the PYTHONPATH variable everything should be fine (the subdirs SPE,sm,etc. aren't needed). Though you must start Blender from a bash – e.g. desktop menus usually don't read the .bashrc/.tcshrc/... and therefore Blender does not know about your user defined environment variables. If you set the PYTHONPATH in /etc/profile instead of .bashrc/.tcshrc/... then starting SPE/SPE from Blender will work also from menus.

Uninstall

Remove any '_spe' and 'sm' folders from your python site-packages directory.:

```
/usr/local/lib/pythonX.X/site-packages
```

One needs also to remove /usr/local/bin/SPE manually.

2.4 Mac Os X

Please follow same instructions as Unix*

So, why there is no specific Mac installer?

This can be solved easily. If you would donate a little more, I could buy a Mac Mini. This will not help only to release, but also to optimize SPE for Mac. A lot of money is there already, so please help to fill the last gap (or donate your old mac). Read more about this fund raising on the SPE homepage.

3 Getting Started

3.1 Startup

Normal mode

Windows

Open the SPE folder and type 'python SPE.py' at the command prompt or make a shortcut to your desktop.

Linux, FreeBSD, Mac Os X, ...

Type 'SPE' on the command line (assuming PREFIX/bin is on your PATH)

Debugging mode

If you have problems starting up SPE, type at the command prompt:

```
> python SPE.py --debug
```

or if you want to report:

```
> python SPE.py --debug > debug.txt 2>&1
```

and send me the error message (debug.txt).

Blender mode

Open SPE.blend and press Alt+P in the corresponding text window.

When SPE is active, the Blender screen will always be redrawn automatically. So the results of any command you type in the interactive shell or of any program you run within SPE, will be visible in the Blender window. Unfortunately it is not possible to interact with Blender directly when SPE is active. So it is impossible to rotate for example the view with the mouse.

3.2 Syntax-checking

Every time you save SPE does syntax checking. If there is any error, SPE will jump to the line in the source code and try to highlight the error.

3.3 Refreshing

SPE has a lot of features like explore tree, index, todo list, and so on... This gets updated every time the file is saved or every time the refresh command is given. This can be done by pressing F5 on the keyboard, the refresh toolbar button or clicking the View>Refresh menu.

3.4 Running files

Warning:

SPE doesn't require you to save your files before running. However it is recommended to do so not to loose source code if your program makes SPE hang.

SPE provides many ways to run files:

Run (F9)

Use this by default, unless you have specific reasons to use the other ones. It will run in the namespace of the interactive shell. So all the objects and functions of your program become available in the shell and in the locals browser (the tab next to the shell).

Run with profile (Ctrl-P)

Same as above but with a profile added. A profile is a report of the program execution which shows which processes or functions are time consuming. So if you want to speed up your code, you can define the priorities based on this report.

Run in separate namespace (Ctrl-R):

Like run, but all the objects and functions defined by the program will not become available in the namespace of the interactive shell. Instead they will be defined in the dictionary 'namespace' of the interactive shell. So if the file 'script.py' is run in this way, type namespace['script.py'] in the shell, to access this dictionary, or namespace['script.py'].keys() to get a list of all defined names, or namespace['script.py'].items() to get tuples of all the names and their values. More easy is to just browse 'namespace' in the locals browser.

Run verbose (Alt-R)

This is for very simple programs, which do not indent more than once. It will send all source lines, as if they were typed in the interactive shell. It is probably a good learning tool for beginners.

Import (F10)

Imports the source file as a module. For running files, they don't have to be saved. For importing files, it is recommended to save them first.

3.5 Separators

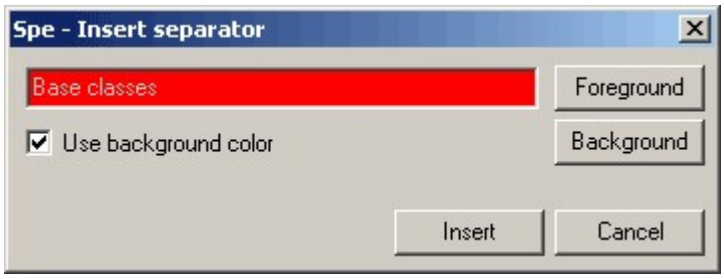
A separator is a label which appears in the explore tree of the sidebar to help structuring the script. An easy way to add separators is to use the 'Edit'>'Insert colored separator'wizard from the menu.

Syntax:

- normal: `#---label`
- colored: `#---label---#foreground color#background color`
- highlighted: `####label`

Foreground and background color are in html notation, eg.:

- red on blue label: `#---red on blue---#FF0000#0000FF`



3.6 Remember option

This can be activated by checking File>Remember or by pressing the heart toolbar button. It will open automatically the scripts which were open in the last session. Useful for Blender if you have to switch continously between Blender and SPE.

3.7 Psyco

If you don't know the python psyco module, you can ignore this item, as it won't have any effect for you. Psyco programs can't run in SPE, as they disable the 'locals()' function. Of course you can edit programs using psyco in SPE, but if you want to run them, comment the psyco activation code out.

3.8 Customize

Keyboard shortcuts

If you want to change the default keyboard shortcuts, open a shortcut file (*.py) _spe/shortcuts/shortcuts.py and adapt it to your own taste. Save it with another name and fill in the name in preferences dialog box, of course without the '.py' extension.

Menus and toolbar (deprecated)

You can define your own menus and toolbar buttons, which can execute any python code and also external files. Look at 'framework/menus/Extra.py' for an example.

Instructions:

1. Suppose you want to add a new menu with the name 'XXX' to the menubar. Create a new file with the name XXX.py in the 'framework/menus/' directory

2. Import or define some actions, with the following structure:

```
def action(script,app,event):
      ...
```

The arguments of the function are:

- `script`: current script window
- `app`: application window
- `event`: event

Some usefull stuff:

- `script.fileName`
- `script.source`
    - `script.source.GetText()`
    - `script.source.SetText(text)`
    - `script.source.GetSelectedText()`
    - `script.source.ReplaceSelection(text)`
- `app.run(fileName)`
  runs an external file
- `app.new()`
  creates a new file
- `app.open(fileName,lineno,col)`
  opens a file at given position
- `app.message(text)`
  shows a dialog window with the text
- `app.messageEntry(text)`
  shows a dialog prompting an entry
- `app.messageError(text)`
  shows an error dialog window with the text
- `app.SetStatus(text)`
  sets the status text

3. Define the 'main' function:

```
def main(app):
       menu(app,
              item(label=<str:label that will appear in menu>,
                     action=<function:that will be called>),
              item(...),
              SEPARATOR,
              item(...),
              item(...),
              ...
              SEPARATOR,
              item(...),
              ...)
```

If you want this menu item also to have a toolbar button, than make a 16x16pixels png image (transparency is allowed). The image has to be located in the menu folder. Pass the the fileName with the toolbar keyword:

```
item(label=<str:label that will appear in menu>,
       action=<function:that will be called>,
       toolbar=<str:fileName of the toolbar image (optional)>)
```

4 Features

4.1 Sidebar

- Class browser
- File browser
- Automatic todo list, highlighting the most important ones
- Automatic alphabetic source index of classes and methods
- Sticky notes

4.2 Tools

These tools appear as tabs down.

Shell

Interactive PyCrust shell

- Double mouse click to jump to error source code

Locals

Local object browser

- Left mouse click to open
- Right mouse click to run

Session

Separate session recorder

Find

Find recursively text in files

- Leave the 'Path' field empty to search in all open files

Browser

Quick access to python files in specified folders and their sub folders

- Left mouse click to open

Recent

Unlimited recent file list

- Left mouse click to open
- Right mouse click to run

Todo

Automatic todo list of all open files, highlighting the most important ones (jump to source)

- Left mouse click to jump to source

Index

Automatic alphabetic index of all open files (jump to source)

Notes

Sticky note for general development comments

Blender

Blender object browser. It is only working when SPE is launched in Blender mode.

Donate

If you like SPE, please consider to give a donation

4.3 Editor

As you type:

- Syntax-coloring
- Auto-indentation
- Auto-completion
- Call-tips

When you save a file:

- Syntax-checking

Uml view

- Graphical layout of class hierarchy

Special keyboard shortcuts

- Ctrl+Enter: browse source of module

4.4 Drag&Drop

Drag&drop any amount of files or folders on ...

- main frame to open them
- shell to run them
- recent files to add them
- browser to add folders

4.5 General

- Context help defined everywhere
- Add your own menus and toolbar buttons
- Exit & remember:all open files will next time automatically be loaded
    - handy for Blender sessions
    - heart icon on toolbar
- Scripts can be executed in different ways: run, run verbose and import

4.6 Blender

- Redraw the Blender screen on idle (no blackout)
- Blender object tree browser (cameras,objects,lamps,...)
- Add your favorite scripts to the menu
- 100% Blender compatible: can run within Blender, so all previous described features are available within Blender

4.7 Windows

- SPE registers itself in the windows explorer context menu
- optional creation of desktop and quick launch shortcuts

5 Tutorial

5.1 Introduction

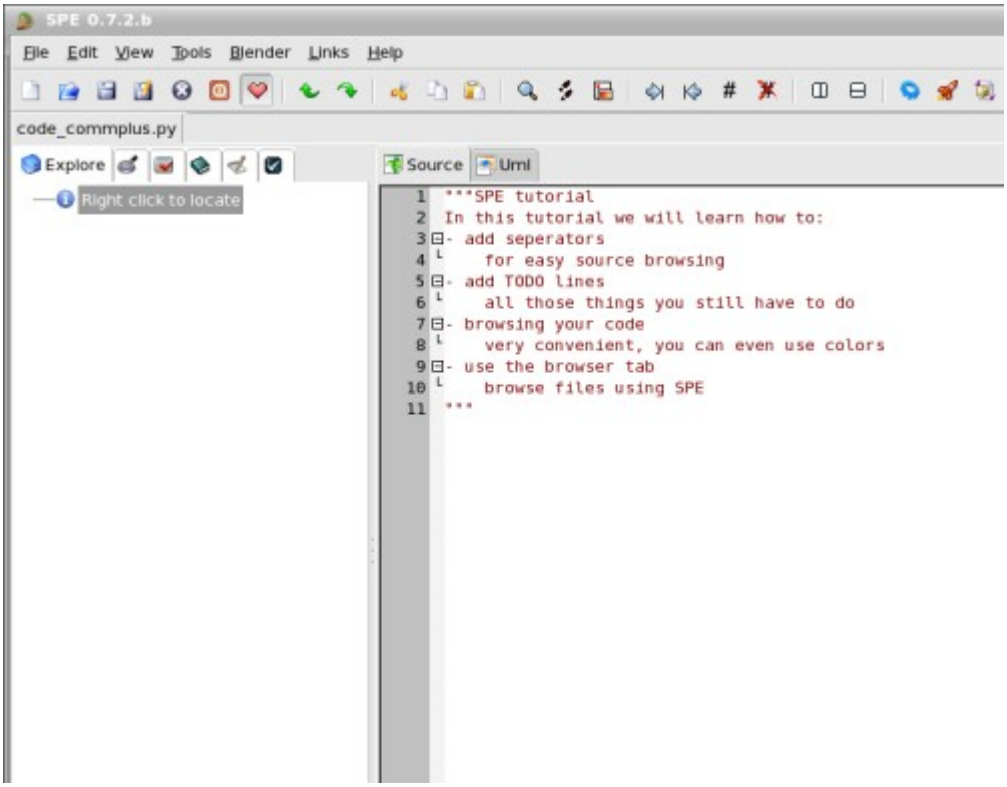
This excellent tutorial was written by Dimitri from www.serpia.com Please visit his website for more Python tutorials.

The tutorial will focus primarily on the functionality of SPE's sidebar. SPE allows you, amongst many other things like syntax coloring, to create seperators which makes it very easy to keep your code structured. This results in a clear and fast way to maintain your code more easily. So I won't tell you anything about SPE's blender support, if there is anyone who wants to write a tutorial on that subject (or any other subject regarding SPE), I'd be more than happy to add it to this webpage.

5.2 The comments

Let's start SPE and study the screen. SPE's main window is roughly divided into four parts. The upper part is where the filemenu and toolbar resides, beneath it you will see two vertical windows, the left one is the sidebar and on the right you'll see the editor itself. On the bottom of the main window is the Python shell and clicking on one of the tabs will give you another view of you code. Some of these tabs are common to the one in the sidebar, but here you will find some extra functionality like a search function and an interface to Blender. As I said earlier, we will primarily focus on the sidebar.

First, add some comments for your source (something you should always do) starting on the first line of the editor. Something like depicted below:



code:

```
"""SPE tutorial
In this tutorial we will learn how to:
- add seperators
    for easy source browsing
- add TODO lines
    all those things you still have to do
- browsing your code
    very convenient, you can even use colors
- use the browser tab
    browse files using SPE
"""
```

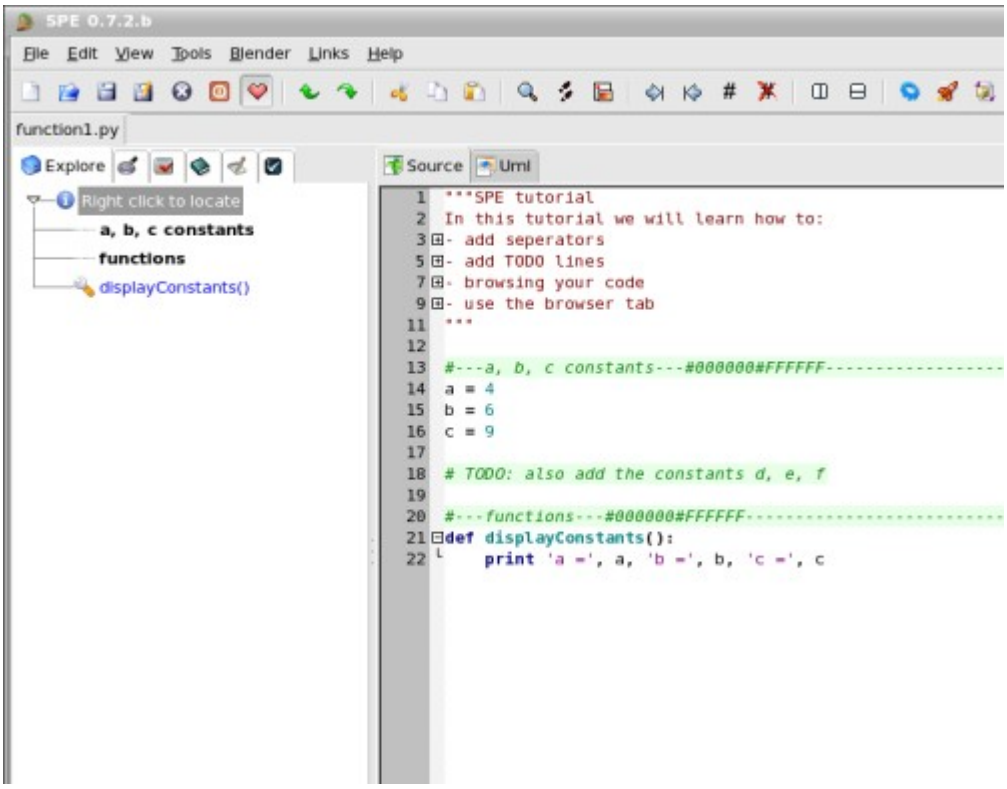
It is very convenient that the text can be placed inside a tree hierarchy, you can expand the text using '+' and vice versa. To give a line a lower hierarchy, press the <tab> key.

5.3 Adding a separator and todo

Adding a separator is a convenient way to structure your code, thus simpler to maintain. This is not only important for large files, you will find that it is also a great feature for smaller files (small files tend to grow bigger). You can add a separator using different methods: just type "#---some text" on an empty line in the editor, select it from the Edit menu or use the Alt+i shortcut. The newly created separator will appear in the explore tab of the sidebars. Rightclicking on the reference to the separator in the explore tab will locate it in your code. This allows you to quickly find chunks of code in your source without having to scroll up and down and staring at the screen. This is, from a view of usability, something that can actually increase your productivity.

Another handy feature is the auto creation of a todo list. Just add '# TODO:sometext' to your code and the todo tab of the sidebar will store the text following the '#TODO:' tag. A very easy way to keep track of the inevitable todo's! But of course, you can store all sorts of other information here for future references (e.g. 'this code fragment is from Harry's webtutorial'). A cool feature of the todo tag is that you can determine its priority by the amount of exclamation marks ("!"). The one with the most exclamation marks will be highlighted. As this you don't have to think any more about the order in which you insert your todo's. There is a special tab dedicated to the todo's on the sidebar, here you can see the priorities of your todo's.

In the next picture you can see that I also added a function definition, this will also appear in the sidebar. The reference for this function in the sidebar uses a blue font and you can use it to jump to the location in your source code. SPE's author was smart enough to add an icon also, human beings are visually orientated and icons work very well in this regard.



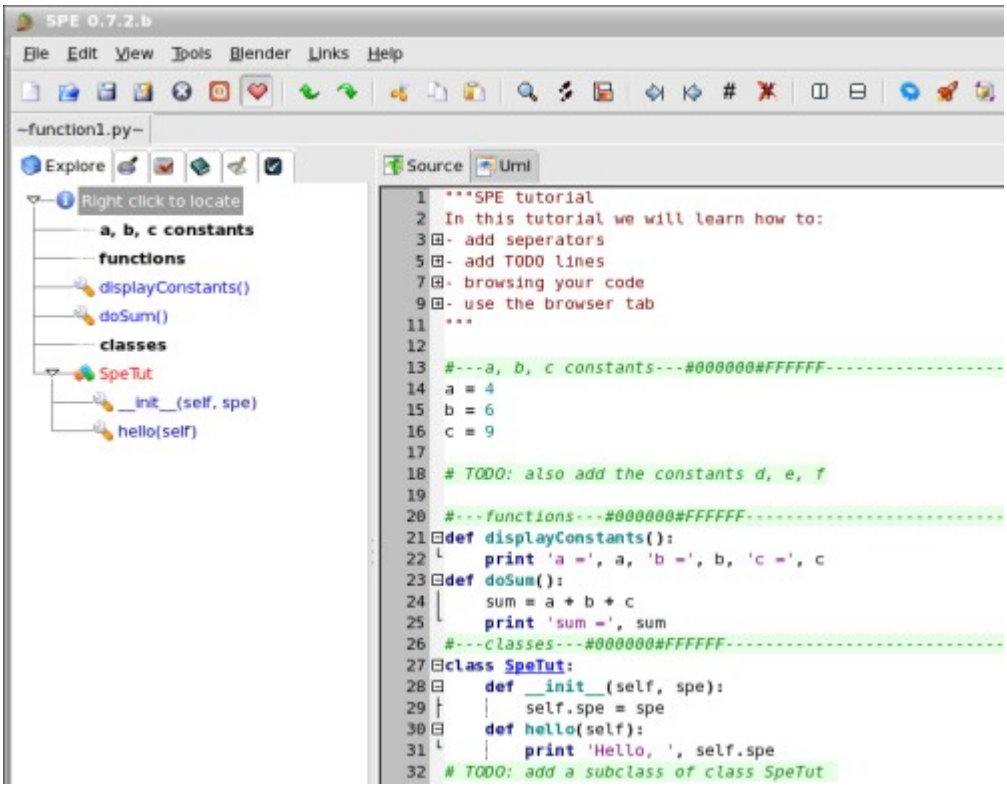
code: function1

```
"""SPE tutorial
In this tutorial we will learn how to:
- add seperators
    for easy source browsing
- add TODO lines
    all those things you still have to do
- browsing your code
    very convenient, you can even use colors
- use the browser tab
    browse files using SPE
"""

#---a, b, c constants---#000000#FFFFFF-----------------------------------
a = 4
b = 6
c = 9

# TODO: also add the constants d, e, f

#---functions---#000000#FFFFFF-------------------------------------------
def displayConstants():
    print 'a =', a, 'b =', b, 'c =', c
```

5.4 Browsing a class

The next thing we'll do is to add a class to our program, but first let's create another separator named "classes". By doing so, it will be easier to identify the location of your classes. Beneath this separator we will create a class named *SpeTut*, this class contains two methods, *__init__* (aka the constructor) and the method *hello*. As you can see in the picture below, the class browser *SpeTut* will be visible in the explore tab, including the aforementioned methods. Use the little triangle on the leftside to expand the tree and vice versa. The reference to the class in the explore tab has a red font and a unique icon (click on it to expand the tree!). You can also add a separator inside a class, a nice feature for larger classes with many methods.

One note though, after you have added a separator, a function or the like you have to refresh the tree by returning it to the highest hierarchy using the triangle.



code: 2func_class

```
"""SPE tutorial
In this tutorial we will learn how to:
- add seperators
    for easy source browsing
- add TODO lines
    all those things you still have to do
- browsing your code
    very convenient, you can even use colors
- use the browser tab
    browse files using SPE
"""

#---a, b, c constants---#000000#FFFFFF-----------------------------------------
a = 4
b = 6
c = 9

# TODO: also add the constants d, e, f

#---functions---#000000#FFFFFF-------------------------------------------------
def displayConstants():
    print 'a =', a, 'b =', b, 'c =', c
def doSum():
    sum = a + b + c
    print 'sum =', sum
#---classes---#000000#FFFFFF---------------------------------------------------
class SpeTut:
    def __init__(self, spe):
        self.spe = spe
    def hello(self):
        print 'Hello, ', self.spe
# TODO: add a subclass of class SpeTut
```

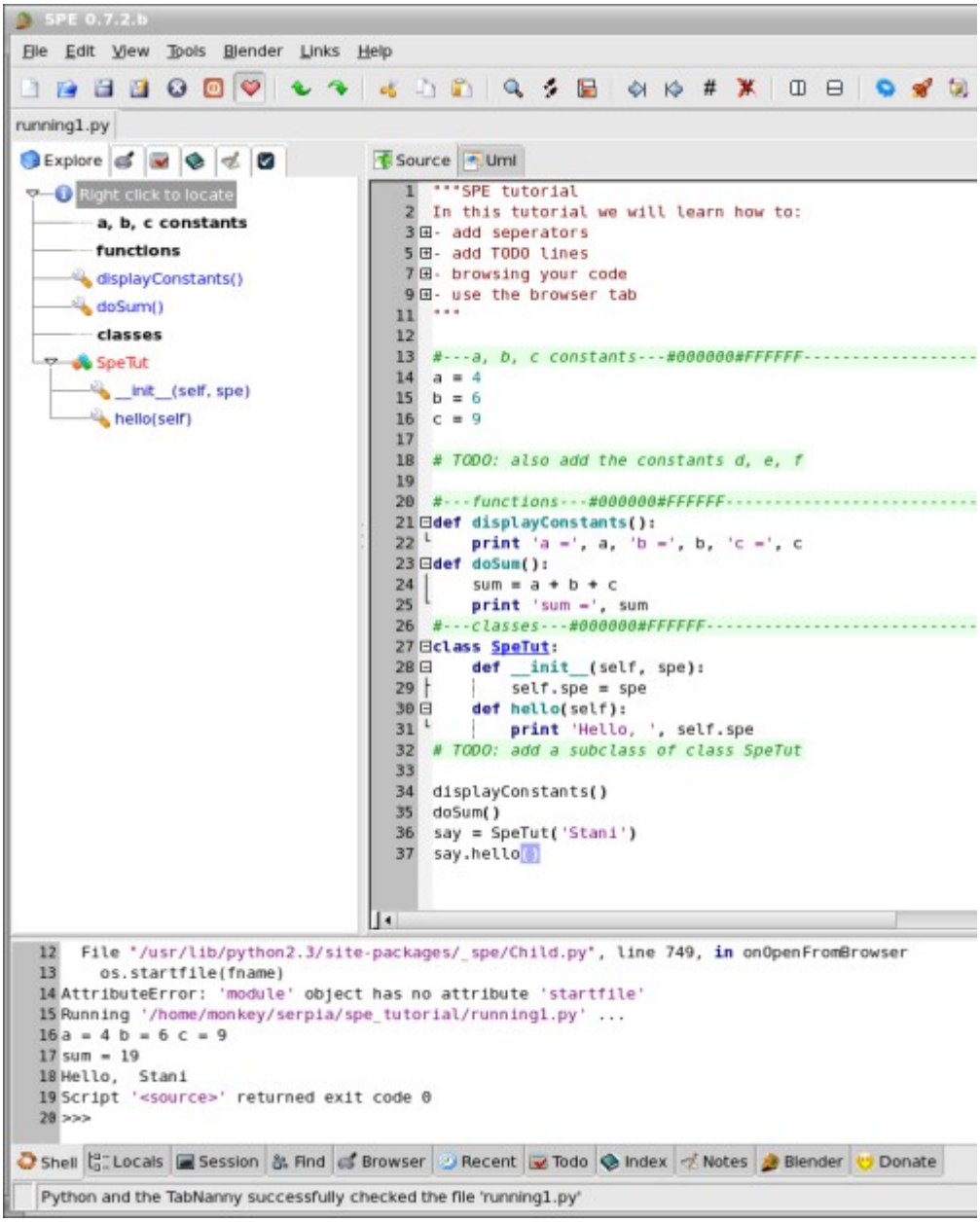
5.5 Run that py!

Now it's time to run our little program, so add:

- displayConstant() [this runs the function definition]
- doSum() [this runs the function definition]
- say = SpeTut('Stani') [this creates an *instance* with an *argument*]
- say.hello() [this runs the method from the class]

to the source code

Use F9 or use the icon on the toolbar and the code will be executed. You can see the output in the Python shell in the lower area of SPE's main window. It's a good practice to do this often as you're building your code. Another nice feature of the sidebar is the source code checker (PyChecker). You can use it by clicking on the appropriate tab located in the sidebar.

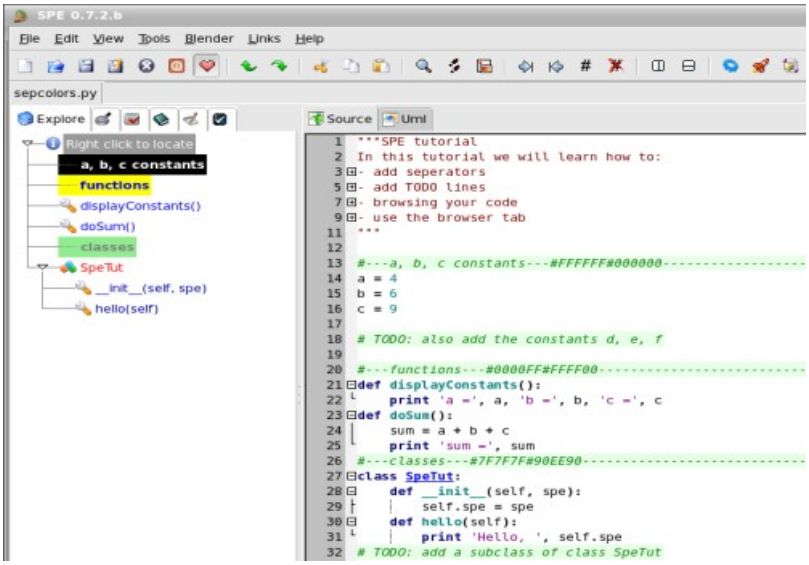


code: running1

```
"""SPE tutorial
In this tutorial we will learn how to:
- add seperators
    for easy source browsing
- add TODO lines
    all those things you still have to do
- browsing your code
    very convenient, you can even use colors
- use the browser tab
    browse files using SPE
"""

#---a, b, c constants---#000000#FFFFFF-----------------------------------------
a = 4
b = 6
c = 9

# TODO: also add the constants d, e, f

#---functions---#000000#FFFFFF-------------------------------------------------
def displayConstants():
    print 'a =', a, 'b =', b, 'c =', c
def doSum():
    sum = a + b + c
    print 'sum =', sum
#---classes---#000000#FFFFFF---------------------------------------------------
class SpeTut:
    def __init__(self, spe):
        self.spe = spe
    def hello(self):
        print 'Hello, ', self.spe
# TODO: add a subclass of class SpeTut

displayConstants()
doSum()
say = SpeTut('Stani')
say.hello()
```

5.6 Life is full of colors

Adding colors to the separator makes it even easier to keep track of your code (as long as you don't turn it into a Christmas tree...). There are two ways to add colors to the separator, a convenient way is to use filemenu --> edit, another way is to type the colorcode (Hex, e.g. #7F7F7F). Use whatever suits you best.

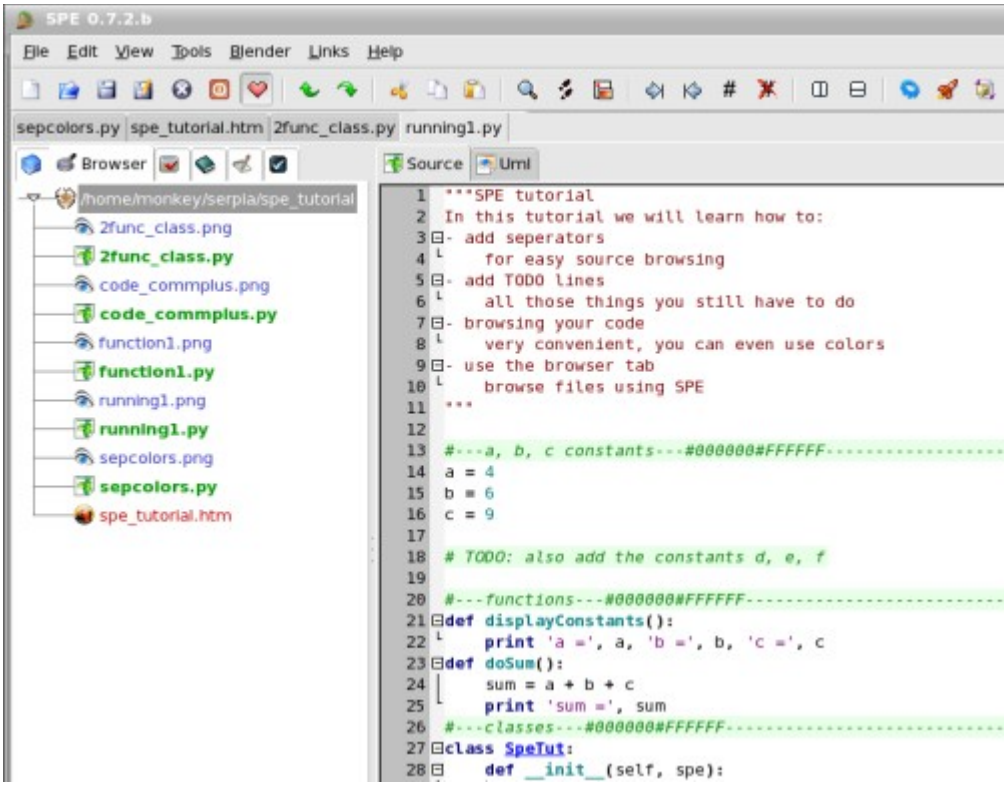


code: sepcolors

```
"""SPE tutorial
In this tutorial we will learn how to:
- add seperators
    for easy source browsing
- add TODO lines
    all those things you still have to do
- browsing your code
    very convenient, you can even use colors
- use the browser tab
    browse files using SPE
"""

#---a, b, c constants---#FFFFFF#000000-----------------------------------
a = 4
b = 6
c = 9

# TODO: also add the constants d, e, f

#---functions---#0000FF#FFFF00-------------------------------------------
def displayConstants():
    print 'a =', a, 'b =', b, 'c =', c
def doSum():
    sum = a + b + c
    print 'sum =', sum
#---classes---#7F7F7F#90EE90---------------------------------------------
class SpeTut:
    def __init__(self, spe):
        self.spe = spe
    def hello(self):
        print 'Hello, ', self.spe
# TODO: add a subclass of class SpeTut

displayConstants()
doSum()
say = SpeTut('Stani')
say.hello()
```

5.7 Browsing your files

Working on a project often means that you have a lot of files that you have keep track of. A recent feature makes it quite easy to do this. Just click on the browser tab and the files of your current directory will be displayed, right-click on a file and it is opened in SPE and you can edit the file. This works for Python files (yeah, right...), but you can also edit html files.



One last feature I will mention in this tutorial is the ability to create sticky notes. Just click on the notes tab located in the sidebar and type some notes about your program. Making notes about your program is more important then you might think, an idea you have today may be forgotten the next day (or the next hour), it's just a small effort to make notes and SPE makes this very easy for you. The notes will be saved as an external '.txt' file and has the same name as your file. Another simple but effective way to keep track of your coding. Once you make this empty, the external file will also disappear.

5.8 The end

Here is where my little tutorial ends (the second version anyway) and I just barely scratched the surface of SPE's functionality. If you are looking for a free Python IDE, you owe it to yourself to try SPE and I think you won't be disappointed. Okay, it lacks a full blown debugger that some *commercial* IDE's have, but the dynamic nature of Python makes it quite easy to do your "own" debugging. Oh, did I tell you that SPE includes wxGlade? You can find a tutorial on wxGlade here!

If you have found any errors or want some extra stuff explained in this tutorial (this is the second version after all), please contact me.

6 wxGlade GUI Designer

6.1 Introduction

This excellent tutorial was written by Dimitri from www.serpia.com Please visit his website for more Python tutorials.

First, what is wxPython?

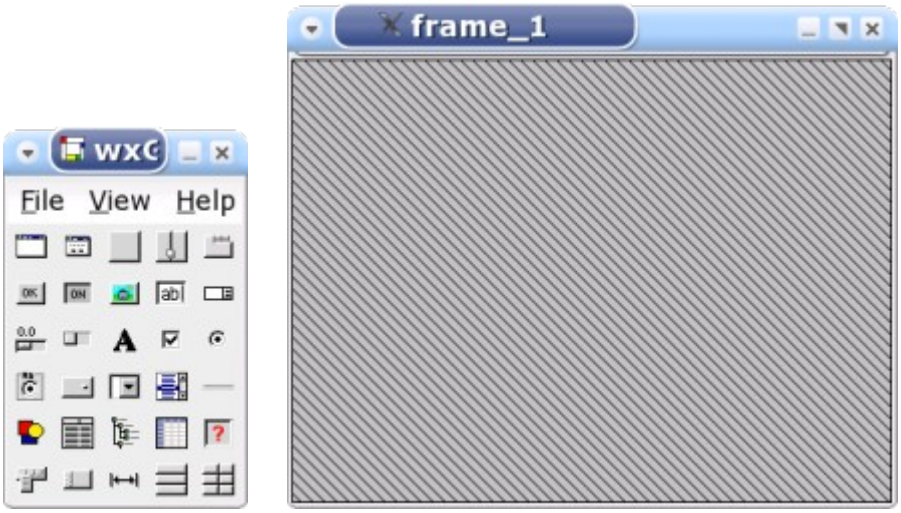
wxPython is a toolkit for creating graphical user interfaces (GUI) for the programming language Python. One of the great things about wxPython is the cross platform compatibility. This means that the same code runs on your Linux, Unix, Windows and Mac OSX box without any problems, and it still has a native look to it. You can find more information about wxPython and download it here. And like all good things in this world, it is Open Source.

What is wxGlade?

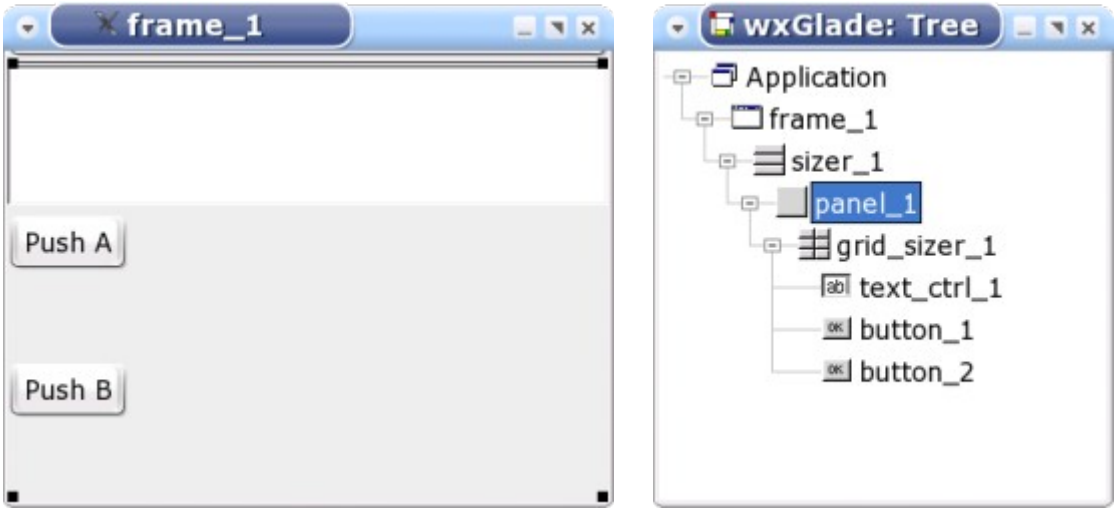
wxGlade is a GUI designer for wxPython. It makes it very easy to create a user interface using drag and drop techniques without writing a single line of code. Anyone who has ever written wxPython code will understand that you can save a lot of time using a GUI designer such as wxGlade. wxGlade also generates C++, Perl and XRC (wxWidgets' XML resources) code. More information about this great piece of software here.

6.2 Design a layout

I assume you had no problems installing wxPython and/or wxGlade, so now it's time to start wxGlade. Click on the button "Add a frame" in the wxGlade's main window and choose for wxFrame, as a result a Frame will be created. wxGlade automatically adds a sizer to the frame, a sizer is a container for our widgets like buttons, etcetera.

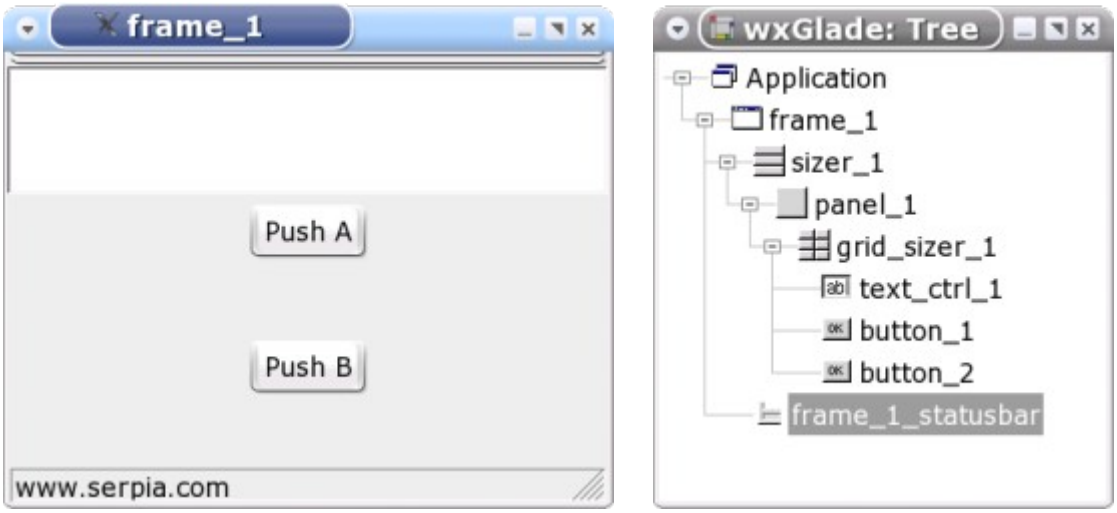


It is very important to add a panel to the sizer, otherwise your application will look funny in Windows. So please add a sizer with 3 rows. Now we will add two buttons and one text field to the frame. Just click on the "Add TextCtrl" in the main window and than click on the upper row in your panel. Repeat this for both buttons. Now you should have something like this:



Time to save our work! Saving is done by "File" --> "Save as..." in the main window, should not be difficult if you've come this far.

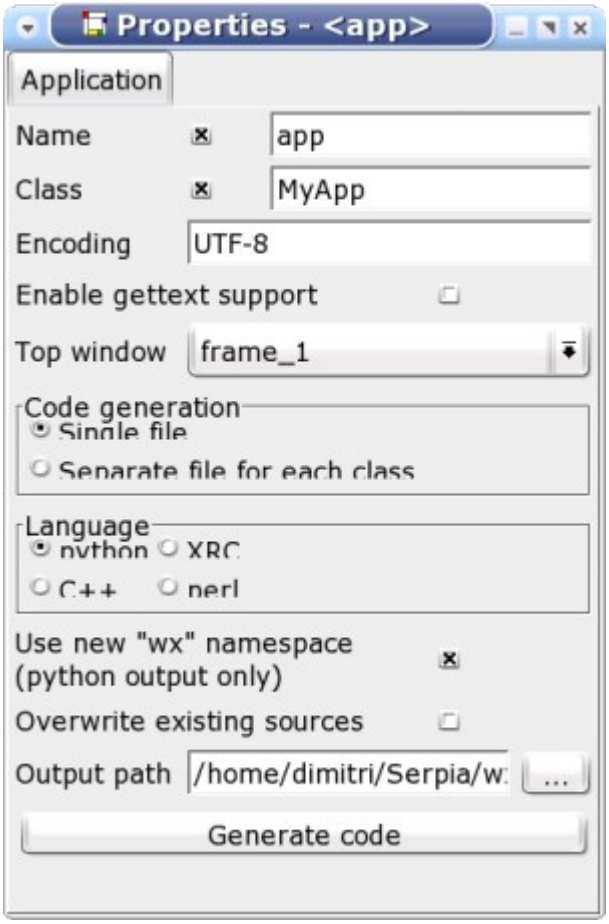
When you click on a widget (e.g. text_ctrl_1) in your frame, its properties will be displayed in the "Properties" window. Now edit the properties of you widgets like I did and you should have something like this:



Hey, did you notice that I added a status bar? Find out how I did it yourself...

6.3 Generate Python Code

Generating Python code is also easy, just keep some things in mind. Click on "Application" in the "Tree" window and the "Properties" window appears. Now, look at the picture below and I'll explain it to you.



Check "Name" and "App" to create a class and mainloop in your code. Select "frame_1" as the Top window. Generate a single file for this application, a more complex application would require separate files, at least this would be recommendable. Check the "Python" radiobutton, select "wx namespace" and do NOT overwrite existing sources (I'll explain this later). Set the output path and filename and "Generate code".

Open this file in your favorite editor and examine the code. It should run now, your wxPython program without having to write a single line of code. Isn't it amazing?

But if you click the buttons, nothing happens. That's because there are no *events* in this program yet. I will explain below how to add events to this little application. Adding events Adding events should be done "by hand", wxGlade cannot do this because it is a GUI builder and not an IDE. So you will need an editor or IDE, such as WingIDE, to add events and other lines of code. It is VERY important to add your code outside the *# begin wxGlade # end wxGlade* section! That is because whenever wxGlade (re)generates a new version (because you have added another button, for example) it leaves your code outside the *# begin wxGlade # end wxGlade* section intact. And that's why you have to uncheck "Overwrite existing sources" in the properties window!

6.4 Event handling

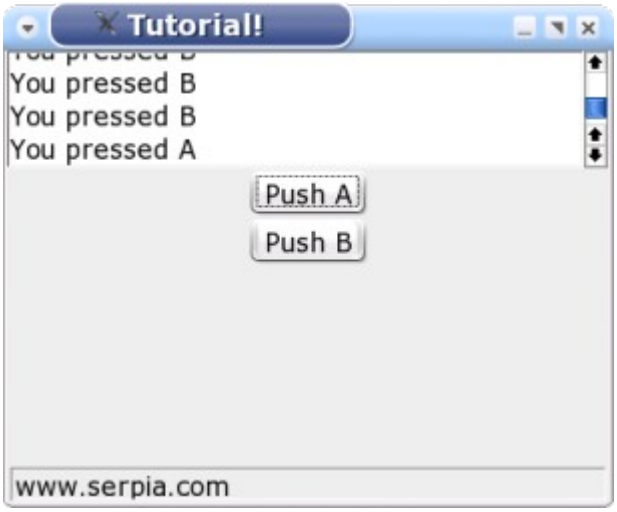
Now fire up your editor and look for the *# end wxGlade* line in the def __init__ section ot he class MyFrame. Below this line you should add:

```
wx.EVT_BUTTON(self,self.button_1.GetId(), self.pushA)
wx.EVT_BUTTON(self,self.button_2.GetId(), self.pushB)
```

This is where the events bindings to the buttons 1 and 2 take place. The self.button_1.GetId() code sees to it that each buttons has a unique ID (well, actually it is a bit more complicated). The self.pushA code is there to let the button know what event should be executed. When you run this code, you will get an error message like "*AttributeError: MyFrame has no attribute pushA*". That is because we haven't defined the pushA and pushB definitions yet. So let's add the following *outside* the def __init__:

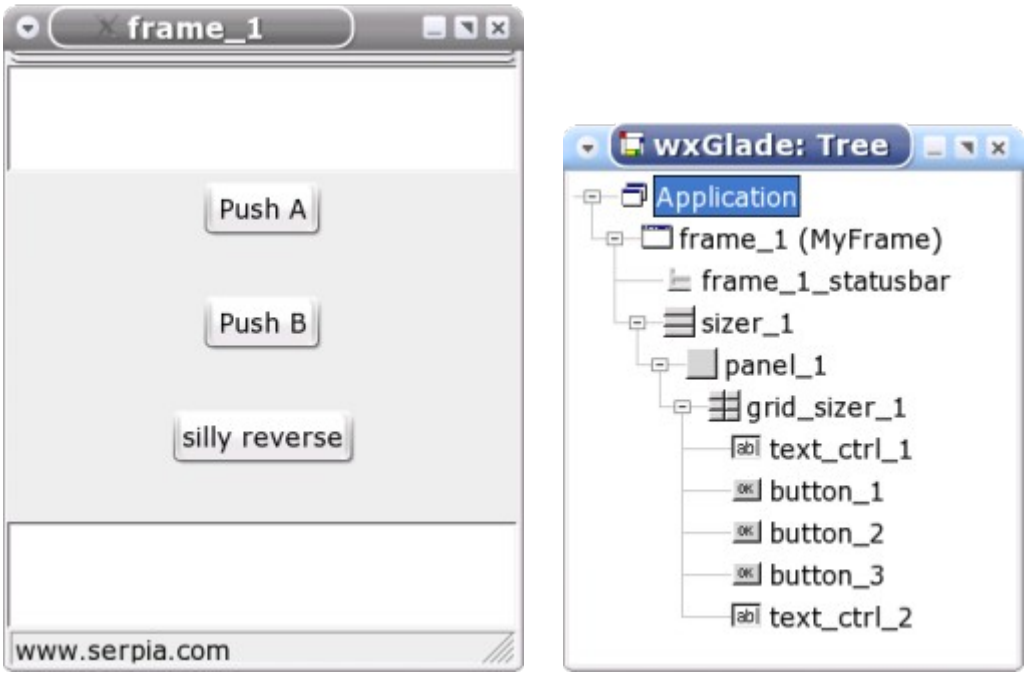
```
def pushA(self, event):
    self.text_ctrl_1.WriteText("You pressed A\n")
def pushB(self, event):
    self.text_ctrl_1.WriteText("You pressed B\n")
```

Both function definitions (*def*) are very simple and a good example of the straightforward logic of wxPython. The text "You pressed ..." is written to (*WriteText*) the *text_ctrl_1* widget we created earlier. By the way, *\n* is a special character and it means *newline*. And yes, it creates a new line after the text that's been written. Run the program and see what happens. Something like this should be the result:



6.5 Now let's enhance this program a bit!

We will add another button and another text_ctrl. First we need to add two extra rows to the grid sizer. To do this right click on grid_sizer_1 in the Tree window and add two rows, one by one. After that's done add a button to the fourth row and a textctrl to the fifth as depicted in the illustrations below.



Add the following event for the new button:

```
wx.EVT_BUTTON(self, self.button_3.GetId(), self.doSilly)
```

Also, add the following function definition:

```
def doSilly(self, event):
    n = list(self.text_ctrl_1.GetValue())
    n.reverse()
    n = ''.join(n)
    self.text_ctrl_2.WriteText(n)
```

This function reads the content from text_ctrl_1 (GetValue()), turns it into a list, reverses the string and joins the characters. The output is then sent to text_ctrl_2. Pretty silly, but hey, this tutorial is for demonstration purposes only.

Try to understand the code, add more functionality as you learn more. And study the excellent examples that come with the wxPython package. You can view the source code of this tutorial here.
Here is another excellent site on wxPython.

If you have found any errors or want some extra stuff explained in this tutorial (this is the first version after all), please contact me.

7 XRCed GUI Designer

7.1 Introduction

This excellent tutorial was written by Dimitri from www.serpia.com Please visit his website for more Python tutorials.

First, what is XRC?

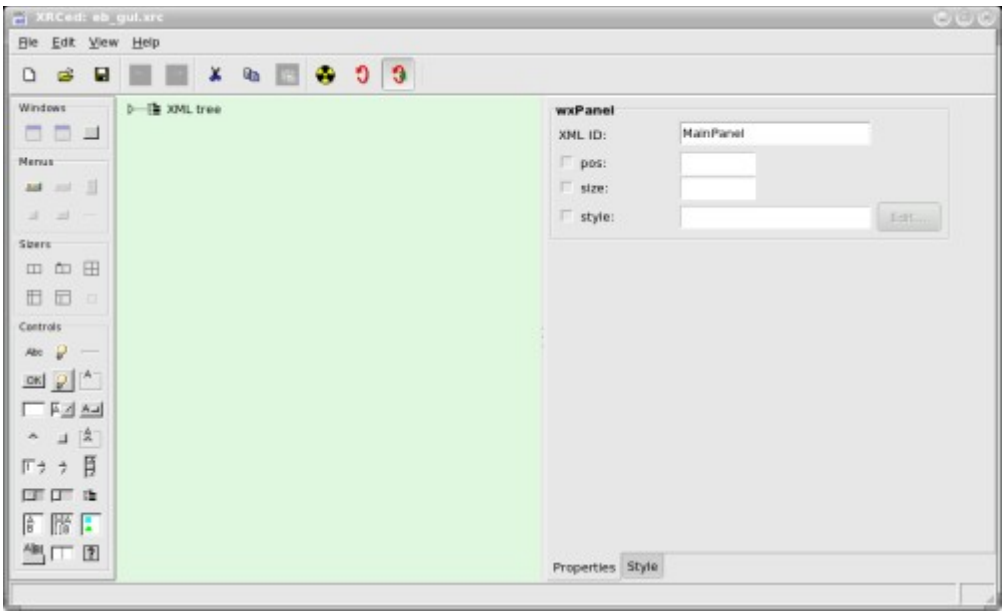
XRC stands for XML Resource Code and it describes the attributes of the widgets used in wxPython. So instead of writing code that builds the widgets, the XML resource file is loaded into the application. There are several advantages to this approach, but the most prominent is the strict seperation between gui design and the functionality of your program.

What is XRCed?

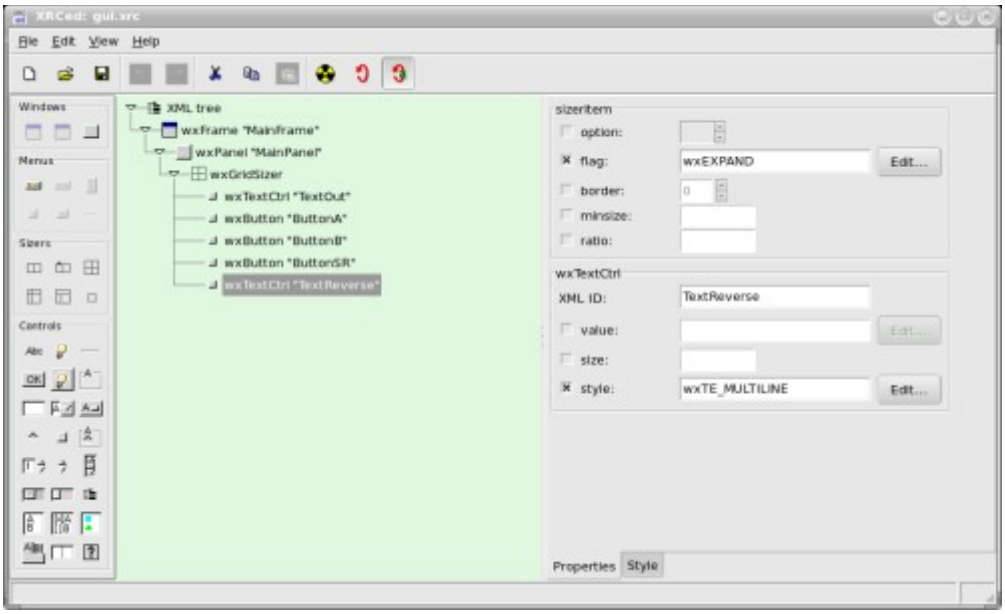
XRCed is is a simple resource editor for wxWindows/wxPython GUI development which supports creating and editing files in XRC format. It is written in Python and uses wxPython GUI toolkit. More information about XRCed here.

7.2 Design a layout

Lets start. I assume you had no problems installing XRCed, so now it's time to run the program. XRCed is also included in SPE (a free, powerful Python IDE), you can find a tutorial on SPE here. The main window of XRCed is depicted below.



Right click on XML-tree and a menu appears, choose "Append child" --> "Frame". After you have added the frame, you can edit the properties and the style of this particular widget. Or any other widget after you have added them. One property of the Frame is very important, the "XML ID". As you will see later in this tutorial, wxPython needs this ID to make it all work. We will name it "MainFrame". Now add a panel the same way as we added the frame and name this panel "MainPanel". After you have added the panel, add a gridsizer, a gridsizer can contain widgets in wxPython and keeps your interface nice and tidy. Try to add more widgets:



view the xml code here and open it in XRCed to see how things work.

As you are building you graphical user interface, you can preview it by double clicking on the MainFrame or by using the icon in the toolbar.
Time to save our work! Save your file as gui.rpc.

7.3 Create your application

Now it's time to embed the xml resource file in your Python code. Start your editor and add the following code:

```
import wx
import wx.xrc as xrc

GUI_FILE_NAME = 'gui.xrc'
GUI_MAINFRAME_NAME = "MainFrame"
GUI_MAINPANEL_NAME = "MainPanel"
GUI_TEXTOUT_NAME = "TextOut"
GUI_TEXTREV_NAME = "TextReverse"
GUI_BUTTONA_NAME = "ButtonA"
GUI_BUTTONB_NAME = "ButtonB"
GUI_BUTTONSR_NAME = "ButtonSR"
```

First, we import wxPython. Next we assign names to the variables GUI_FILE_NAME etc. You seen how they correspond with the XML ID's we created earlier in XRCed? By the way you don't have to create this variables, but it sure makes your code easier to maintain. You can even put them in a seperate file. Now add this:

```
class MyApp(wx.App):
    def OnInit(self):

        self.res = xrc.XmlResource(GUI_FILE_NAME)

        self.frame = self.res.LoadFrame(None, "MainFrame")
        self.panel = xrc.XRCCTRL(self.frame, GUI_MAINPANEL_NAME)
        self.textOut = xrc.XRCCTRL(self.panel, GUI_TEXTOUT_NAME)
        self.textRev = xrc.XRCCTRL(self.panel, GUI_TEXTREV_NAME)

        self.frame.Show(True)
        self.InitMenu()
        return True
```

We create the class MyApp, the 'wxXmlResource' loads the XML resource file that we have created earlier. The 'XRCCTRL' command calls the names from the panel and the textboxes from the XML resource file. We will need them later when we create the events.

```
    def InitMenu(self):
        wx.EVT_BUTTON(self, xrc.XRCID(GUI_BUTTONA_NAME), self.pushA)
        wx.EVT_BUTTON(self, xrc.XRCID(GUI_BUTTONB_NAME), self.pushB)
        wx.EVT_BUTTON(self, xrc.XRCID(GUI_BUTTONSR_NAME), self.doSilly)
```

This is where the events bindings to the buttons place. The XRCID(GUI_BUTTONA_NAME) code sees to it that the properties are loaded from the XML resource file. The self.pushA code is there to let the button know what event should be executed. When you run this code, you will get an error message like "*AttributeError: MyFrame has no attribute pushA*". That is because we haven't defined the pushA and pushB definitions yet. So let's add the following:

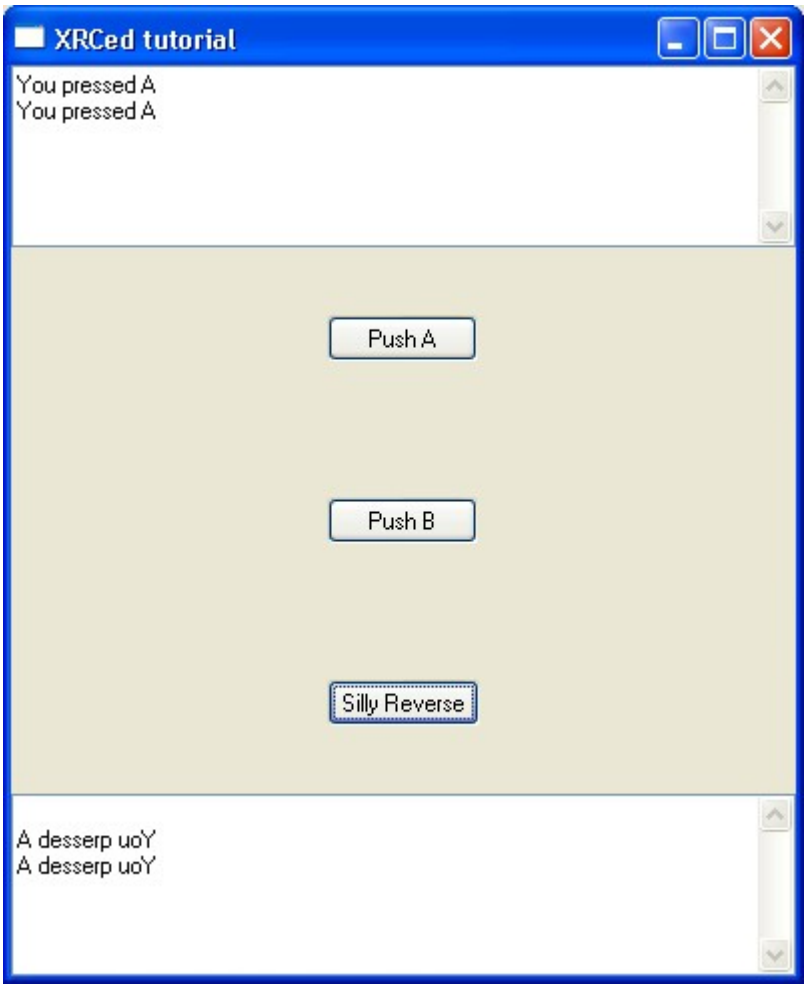
```
    def pushA(self, event):
        self.textOut.WriteText("You pressed A\n")

    def pushB(self, event):
        self.textOut.WriteText("You pressed B\n")

    def doSilly(self, event):
        n = list(self.textOut.GetValue())
        n.reverse()
        n = ''.join(n)
        self.textRev.WriteText(n)
```

Both function definitions (*def*) are very simple and a good example of the straightforward logic of wxPython. The text "You pressed ..." is written to (*WriteText*) the*textOut and textRev* widgets we created earlier. By the way, *n* is a special character and it means *newline*. And yes, it creates a new line after the text that's been written.

To be able to run the program and add:

```
app = MyApp(0)
app.MainLoop()
```

and see what happens. Something like this should be the result:



Try to understand the code, add more functionality as you learn more. And study the excellent examples that come with the wxPython package. You can view the source code of this tutorial here .

Also, compare this tutorial with the wxGlade tutorial on this website and examine the differences between using a XML resource file and writing the code for the gui inside a Python file.

Here is another excellent resource on XRC .

If you have found any errors or want some extra stuff explained in this tutorial (this is the first version after all), please contact me.

8 Debugger

8.1 Introduction

SPE & Debugger

To launch the debugger from SPE, choose "Debug" from the Tools menu in SPE. The python files of the debugger can be found in the "site-packages/_spe/plugins/winpdb" folder. The following documentation is general information about the Python Debugger. For further questions about the debugger, please use its source-forge project page for support requests, bug reports, and forum: http://sourceforge.net/projects/winpdb/

Copyright Notice

Reminder

Except for GUI documentation, whatever is written about Winpdb applies to rpdb2 too. Specifically, whenever _winpdb.py is used in the docs, it can be substituted with _rpdb2.py

8.2 Requirements

wxPython

To use the Winpdb GUI you need to install wxPython 2.6.x
You can still use *_rpdb2.py* which is the console version of the debugger even without wxPython.
http://www.wxpython.org/

Python Cryptography Toolkit

Winpdb uses the Python Cryptography Toolkit to encrypt its socket communication. You can still use Winpdb without the toolkit, but then connections will be authenticated only.
http://www.amk.ca/python/code/crypto

Firewalls

You may experience connectivity problems that stem from firewall protection. Winpdb communicates with debugees over sockets. These sockets require that TCP ports 51000 to 51019 be unblocked to outgoing connections on the debugger machine and to incoming connections on the debugee machine. Usually unblocking TCP port 51000 alone will be enough, unless more than one active debugee is present or port 51000 is occupied by another process.

Multiple Threading

Winpdb requires the presence of the *thread* module.

8.3 Launching and Attaching

So, you have installed Winpdb, what now?

To start the debugger on UNIX systems, open a console and type:

```
_winpdb.py
```

On Windows systems you start the debugger with:

```
python %PYTHONHOME%\Scripts\_winpdb.py
```

Use the *-h* flag for command line help. A common flag at this point is *-t* which allows Winpdb to start even if the Crypto package is not installed.

The above technique starts the debugger, without starting a debug session, in the DETACHED state. You can start a debug session from the command line by appending the debug script name and its command line arguments to the command line of the debugger. For example:

```
_winpdb.py myscript.py myscript_arg1 myscript_arg2 ...
```

This will automatically launch the debugged script when the debugger starts. During this phase the debugger will move from the DETACHED state, through the LAUNCHING, ATTACHING, and BROKEN states. Once the debugger reaches the BROKEN state, it is ready for further commands.

Another option is to start the debugger and launch the script from the debugger console with the *launch* command.

```
launch myscript.py myscript_arg1 myscript_arg2 ...
```

Attaching to a Running Script

Launching starts the debugged script on the local host. What if you want to debug a script on a remote machine? To do that you need to start the debugger on the remote machine in "debugee" mode with the *-d* flag. Example:

```
_winpdb.py -d -r -p"mypassword" myscript.py myscript_arg1 myscript_arg2 ...
```

This will start the debugged script and break into it. At that point the debugged script (debugee) remains frozen until a debugger attaches to it. The *-p* flag sets the connection password and the *-r* flag allows connections from remote machines.

To attach to the debugee start the debugger as follows:

```
_winpdb.py -p"mypassword" -ohostname -a myscript.py
```

This will start the debugger and initiate an attachment attempt to the script *myscript.py* on host *hostname*. Another option is to start the debugger and attach from the debugger console with the *password*, *host*, and *attach* commands as follows:

```
password mypassword
host hostname
attach myscript.py
```

Using the *attach* command without argument will list all scripts available for attachment on the given host.

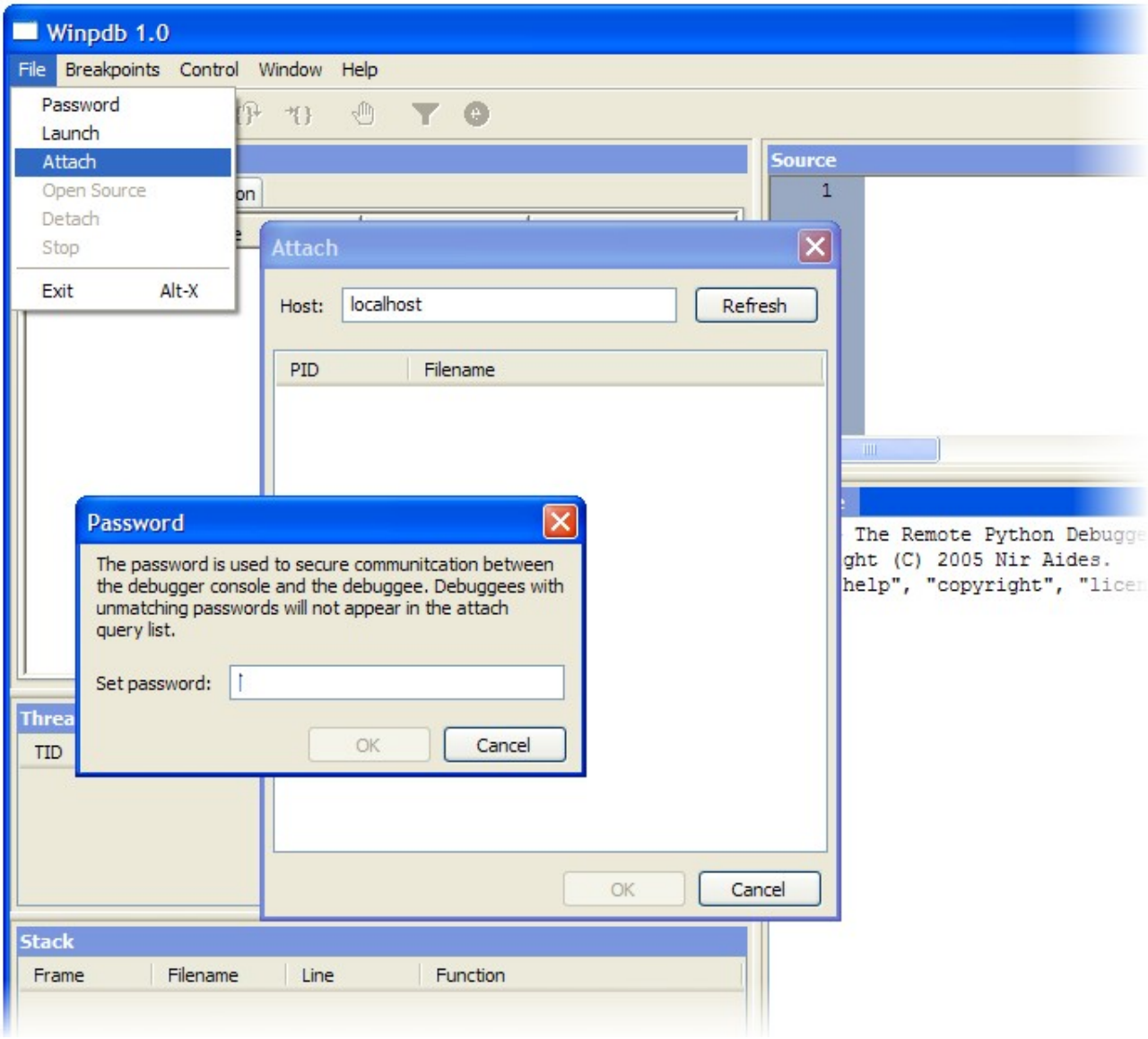
8.4 GUI control:

Setting a Password

To set the connection password click File->Password. This option is available only when the debugger is in the detached state.

Attaching to a Script

To attach to a script click File->Attach. This option is available only when the debugger is in the detached state. If the attach option is selected without a password having been set, the password dialog will pop up automatically, requesting for a password. To attach to a script on a remote machine, type the machine hostname in the *Host* edit box and click the *Refresh* button. Only scripts that match the password will show up in the dialog. Also, note that connectivity issues such as firewall protection may prevent scripts from being detected.



8.5 Embedded Debugging

Normally you would start a debug session by launching a script from the debugger. However, in some scenarios this is not possible. For example python scripts that are embedded in other programs. You can still debug embedded scripts by using the following technique:

Add the following line to any script you wish to debug with the embedded debugging technique:

```
import rpdb2; rpdb2.start_embedded_debugger(password)
```

Once this line is invoked, the script will freeze for a default period of 5 minutes, waiting for a debugger to attach. The password is used to secure client/server (debugger/debugee) communication. Naturally, the debugger has to use the same password in order to successfully attach.

Interactive Passwords

It is recommended not to use a hard coded password in a script, since anyone with read access rights to the script may read the password and compromise your system security. Instead it is preferable to query the password interactively. If applicable you can use the following line as an alternative to the one suggested above:

```
import rpdb2; rpdb2.start_embedded_debugger_interactive_password()
```

A common flag for both functions is *fAllowUnencrypted*, which allows unencrypted connections in case the crypto package is not installed.

Embedded Timeout

What if for any reason you fail to attach to the frozen script? The frozen script waits for you to attach for a default period of 5 minutes, and when this timeout expires it will resume execution. This prevents the need to terminate the server hosting the python script or all kinds of other desperate attempts in the hope of terminating the frozen script.

The functions are brought here for your convenience:

```
def start_embedded_debugger(
                pwd,
                fAllowUnencrypted = False,
                fRemote = False,
                timeout = TIMEOUT_FIVE_MINUTES,
                fDebug = False
                ):

    """
    pwd     - The password that governs security of client/server communication
    fAllowUnencrypted - Allow unencrypted communications. Communication will
                        be authenticated but encrypted only if possible.
    fRemote - Allow debugger consoles on remote machines to connect.
    timeout - Seconds to wait for attachment before giving up. If None,
              never give up. Once the timeout period expires, the debugee will
              resume execution.
    fDebug  - debug output.
    """

    return __start_embedded_debugger(pwd, fAllowUnencrypted, fRemote, timeout,
fDebug)
```

```
def start_embedded_debugger_interactive_password(
            fAllowUnencrypted = False,
            fRemote = False,
            timeout = TIMEOUT_FIVE_MINUTES,
            fDebug = False,
            stdin = sys.stdin,
            stdout = sys.stdout
            ):

    if g_server != None:
        return

    if stdout != None:
        stdout.write('Please type password:')

    pwd = stdin.readline()[:-1]

    return __start_embedded_debugger(pwd, fAllowUnencrypted, fRemote, timeout,
fDebug)
```

8.6 Multiple Threads

Unique Little Beings

While few python debuggers support threading, Winpdb may be the first Python debugger to do it right. Winpdb uses a novel approach to handling threads in the context of a debugger. Python threads are unique little beings. Unlike C++, you can't always break into them (make them stop), since they are not always doing Python code, and may actually be blocked indefinitely in some C++ code. And yet, even more peculiar is the fact that a Python session may exist without any threads of execution at all, for example, think of the python interactive interpreter.

Breaking Into the Debugger

In debugger lingo "breaking into the debugger" means requesting the debugee to pause for our inspection. In Winpdb this is nothing more than a polite request. Individual threads will break at their leisure, and until they do their state is reported as running. The cool thing about the Winpdb model is that we can still control the debugee in this half broken state as if it was completely broken.

The second scenario, in which no threads exist at all when the break is requested, is only relevant to embedded debugging. In that case we can do very little until the first thread shows up on the radar and the debugger finally really breaks.

Threads of the thread module

There are three kinds of threads in Python. The main thread, threads created through the *threading* module, and threads created via the *thread* module. The first two types of threads are traced by Winpdb automatically. However threads created via the *threads* module are born invisible to Winpdb. To make Winpdb trace a thread created with the thread module, add the following line to the thread's function:

```
rpdb2.settrace()
```

Again, this line is only needed for threads created with *thread.start_new_thread()* and is ignored for other kind of threads.

8.7 Smart Breakpoints

Valid Line Breakpoints

Winpdb is the first python debugger that allows you to set breakpoints to valid lines only. In python, some source lines are never executed, so setting a breakpoint to such lines results in the debugger ignoring them. With Winpdb you don't have to guess which lines are valid since the debugger knows that for you, and sets the breakpoint to the nearest previous valid line.

Persistent Breakpoints

Winpdb automatically saves breakpoints when you end a debug session. Next time you start a debug session to the same script, you can load the saved breakpoints. You can even save and load breakpoints manually and have more than one set of breakpoints by specifying a breakpoints file name.

Sticky Breakpoints

Winpdb is the first python debugger that uses truly sticky breakpoints. You can change a script considerably, and yet, next time you debug it and load the saved breakpoints, they will remain in the correct source lines.

Console Commands:

bp – Set a breakpoint.
bl, bd, be, bc – List, disable, enable, and clear breakpoints respectively.
load, save – Load and save breakpoints respectively.

Examples

```
bp 28 - Set a breakpoint to line 28 in the current file.
bp myscript.py:28 - Set to line 28 in file myscript.py.
bp myscript.py:CMyClass.my_method - Set to first line of method my_method of
class CMyClass
bp foo, i > 100 - Set a conditional breakpoint to first line of function foo.
bd * - Disable all breakpoints.
save - Save breakpoints to the default session file.
save my_breakpoint_file - Save breakpoints to a file named
'ny_breakpoint_file.bpl'.
```

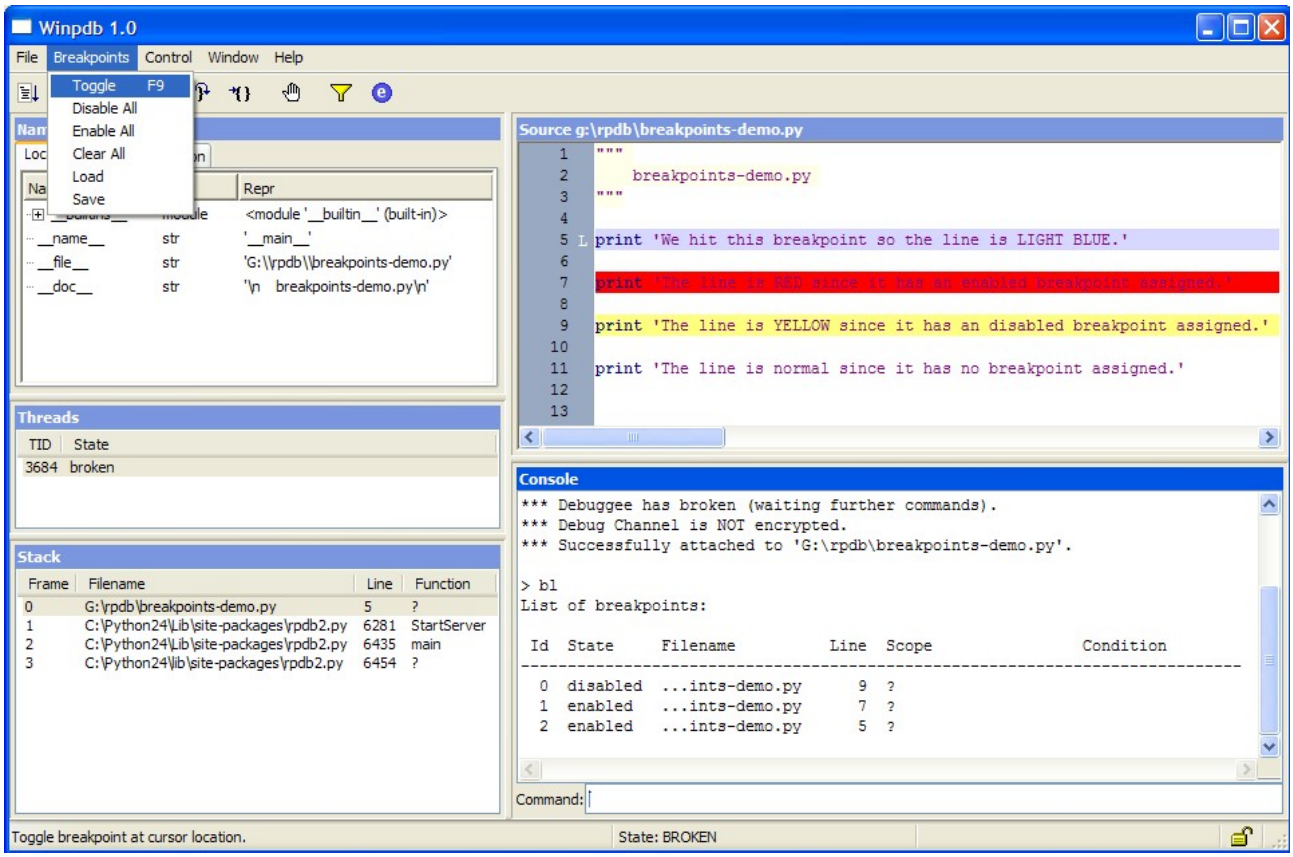
GUI control:

Toggle a Breakpoint

To toggle a breakpoint in the current line in the current file, click Breakpoints->Toggle or F9
Another option is to click the left margin of a source line in the source viewer.

Breakpoint Color

Breakpoints are represented in the source viewer as a colored background for a line with a breakpoint. The colors are RED for an enabled breakpoint and YELLOW for a disable breakpoint. When a breakpoint is hit its color is LIGHT BLUE.



8.8 Security

Authenticated Communication

As a remote debugger Winpdb uses sockets to communicate between the debugger and the debugged script (debugee). This communication is password authenticated, so that an intruder will not be able to control the debugee.

Encrypted Communication

By default the socket communication is also encrypted. Winpdb uses the Python Cryptographic Toolkit (http://www.amk.ca/python/code/crypto) for encryption. Encryption can be allowed off (example: if the Crypto module is not present) with the -t flag.

Automatic Passwords

If a debug session is launched from Winpdb without having set a password, a pseudo random password will be generated transparently, without interrupting the user.

Remote Connections Denial

By default, the debugee denies remote connections, and only accepts debugger connections from the local host. However, the debugee can be set to accept connections from remote machines with the -r flag.

Command Line Flags:

```
-t - Allow unencrypted connections
-p <password> - Set communication password
-r - Allow connections from remote machines.
```

Examples

Start Winpdb in allow unencrypted connections mode. This flag must be set if the Crypto toolkit is NOT installed, either on the debugger or on the debugee machine:

```
_winpdb.py -t
```

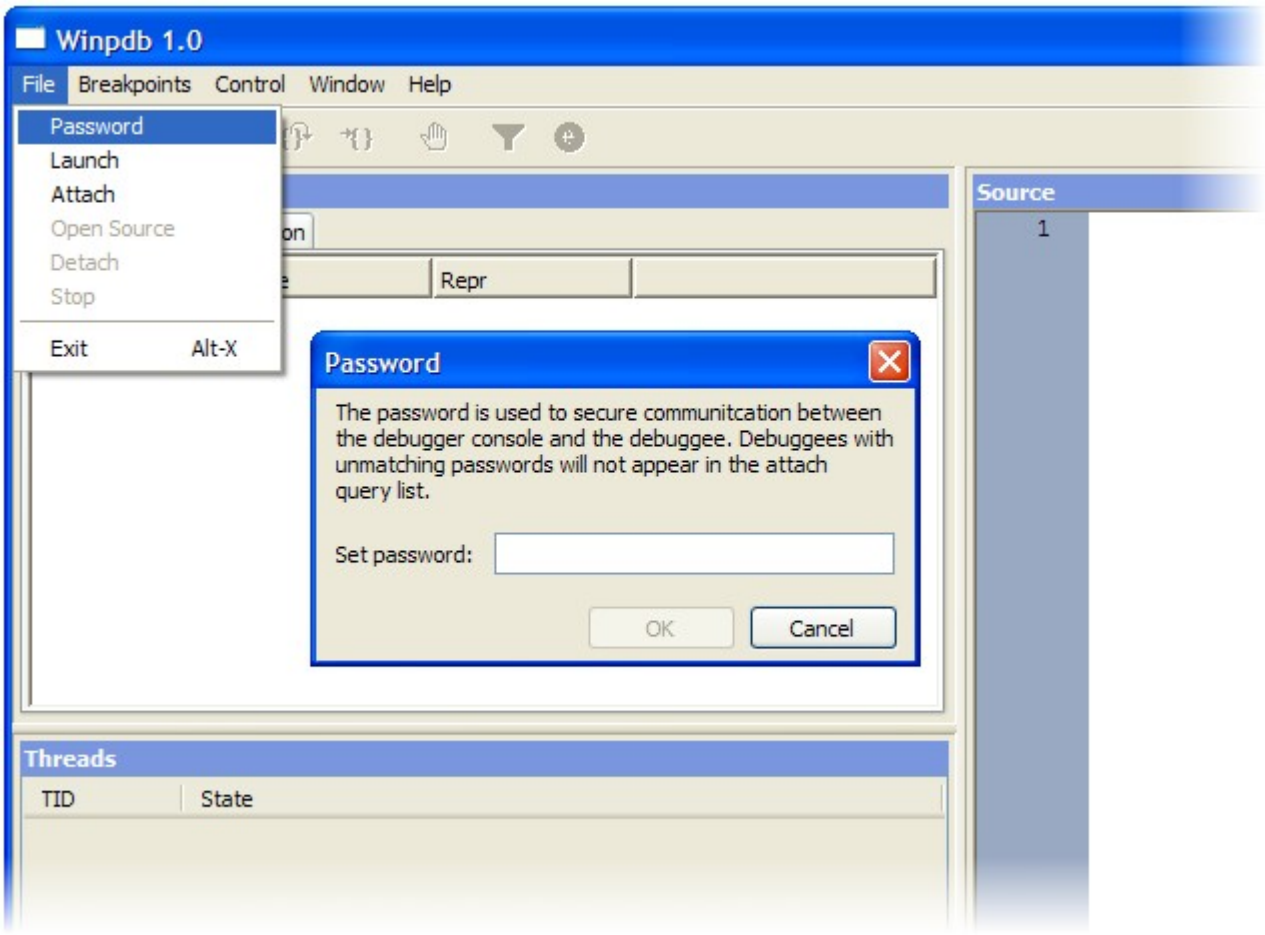
Console Commands:

remote - Get or set the allow remote connections mode.
password - Get or set the password that governs connections.

GUI control:

Set Password

To set the password, click File->Password
If a password has not been set when the attach command is first invoked, the password dialog will pop up automatically.



9 FAQ

9.1 Install

Encoding Error

Q

I'm trying to install SPE 0.7.4.z on my Linux (Ubuntu) box.

I start Python and type "import _spe", but then it tells me:

```
"/usr/lib/python2.4/site-packages/_spe/info.py", line 96, in ?
    INFO['encoding']    = wx.GetDefaultPyEncoding()
```

What more do I need to do to get your IDE up and running?

A

Your version of wxPython is out of date, you need at least wxPython 2.5.4.1 Please upgrade and if you don't know how, trial and error an old version of SPE:

http://projects.blender.org/frs/?group_id=30&release_id=209

Don't ask which one, for sure one will work. You can always try to send an email to your linux distibution to ask to make a new wxPython package available or that they upgrade wxPython in next release.

Ogl Missing

Q

During install I get this ogl missing error....

```
  File "C:\Progs\Python23\Lib\site-packages\sm\uml.py", line 3, in ?
    import wx, wx.lib.ogl as ogl
ImportError: No module named ogl
```

A

Your version of wxPython is not built with ogl support. Again you need at least wxPython 2.5.4.1 with ogl support enabled. The ogl demo in the wxPython demo also won't work for you. This has nothing to do with SPE, so please, if you have further questions about these, please post them on the wxPython-user list:

http://www.wxpython.org/maillist.php

9.2 Editor

Find & Replace Dialog

Q

I have to press twice the “Find Next” button to find the next occurrence of a word.

A

Unfortunately this is a bug in wxPython with unicode support. It also happens in the wxPython demo. It is not related to SPE. If you don't really need unicode support, you could install the ansi version of wxPython as a solution.

Opening Files

Q

When opening an existing file, the X close-button for that file (at the extreme right, on the menu "bar") as well as the minimise/maximise buttons for that file/window disappear.
If many files are open, and one closes one using the round X-button (fifth gui button from the left), the other buttons reappear. This is mainly a cosmetic bug.

A

This is a wxPython bug. It also happens in the wxPython demo. It is not related to SPE.

Saving files

Q

Save a source with special characters like "á" fails. Is this a bug?

A

This is not a bug, your encoding is wrongly configured. Start SPE in debugging mode and check your encoding:

```
>>> python SPE.py --debug
Spe is running in debugging mode with this configuration:
- platform : win32
- python : 2.3.2
- wxPython : 2.6.1.0.
- interface : <default>
- encoding : ascii
```

If it's ascii, you should change it in the preferences dialog box to 'latin_1'. When you restart SPE now in debugging mode, it should display:

```
>>> python SPE.py --debug
Spe is running in debugging mode with this configuration:
- platform : win32
- python : 2.3.2
- wxPython : 2.6.1.0.
- interface : <default>
- encoding : latin_1
```

The problem could also be related to Python default encoding. I had the very same problem, and then I found I had a pre-compiled sitecustomize.pyc and .pyo in my site-packages folder . Just delete any sitecustomize.* file in such a dir of yours, create a new one and put this in:

```
import sys
sys.setdefaultencoding('latin-1')
```

10 Contact

10.1 Contribute

If you would like to contribute to SPE in any way, send me an email with your skills

- programming
- graphics
- icons
- 3d
- html

We are sure you can help us.

10.2 Feedback

SPE is still under development. If you use SPE, please post a message on the appropiate forum on http://projects.blender.org/forum/?groupid=30 describing the platform, the problems that occur and possible solutions if you know.

If SPE runs without any problems, I'm also interested to get a notice.

We develop SPE under Windows XP and have no access to Linux, Mac, FreeBsd or any other platform. So any help for these platforms is highly appreciated.

10.3 Contact persons

These people are contact persons for (replace $ with @):

- Project leader: Stani (spe.stani.be@gmail.com)
- SVN and bugfixes: Sam Widmer (rigel$asylumwear.com)

11 Donations and sponsorship

11.1 Donations

Please donate if you enjoy using SPE and would like to help support it. Your donation will go directly towards helping this project. Any donation starting from 5 euro/dollar is welcome. If you know any fund which would be helpful, please let me know. Large donations can be rewarded with a link on the SPE website or name mentioning in SPE documentation.

We offer four easy ways to make a donation to SPE:

Bank Transfer (Europe)

We strongly recommended this payment for europe as no payment fees are involved. The Dutch Rabobank accepts international transfers. Using the IBAN number, this transaction is free of charge within Europe. So what you donate, is what SPE gets.

at the name of:

S. Michiels, Amsterdam, the Netherlands

Bank: Rabobank

IBAN: NL12 RABO 0393 8648 47 (for euro countries)

Swift/BIC code: RABONL2U (international code)

Account number: 3938.64.847

PayPal (International)

If you have a major credit card (Visa, MasterCard, American Express) or a PayPal account, donating is easy:

- Just click the button on the SPE website to get started
- You can pay through the PayPal site (http://www.paypal.com) to s_t_a_n_i$yahoo.com (replace '$' with '@').

Google Adsense

If you have a website, you can put Google ads on your site which will give me some income, without that you have to pay anything. Please contact me for more information spe.stani.be@gmail.com.

11.2 Sponsorship

Your organisation may sponsor SPE for one or more of the following reasons:

- Helping foster the growth of SPE
- Increasing brand recognition among Python community in specific and open source community in general

Sponsors

Silver

 http://www.zettai.net

Packages

SPE offers three packages for sponsoring:

Platinum: €2000/year

- One available
- Company logo placement and link on SPE website as platinum sponsor
- Company logo placement and link on SPE documentation as platinum sponsor
- Company link on SPE mailing lists as platinum sponsor
- Company link on SPE release announcements as platinum sponsor (comp.lang.python, comp.lang.python.announce, pypi, blender.org, ...)

Gold: €1000/year

- Four available
- Company logo placement and link on SPE website as gold sponsor
- Company logo placement and link on SPE documentation as gold sponsor
- Company link on SPE mailing lists as gold sponsor

Silver: €500/year

- Unlimited availability
- Company logo placement and link on SPE website as silver sponsor
- Company logo placement and link on SPE documentation as silver sponsor

How to apply

If you would like to sponsor SPE in one of ways mentioned above, please send an e-mail with subject "Premium", "Gold" or "Silver" to spe.stani.be@gmail.com

12 Keyboard shortcuts

Key	Action	Description
ALT '3'		Comment
ALT '4'		Uncomment
ALT 'D'	DEDENT	Dedent the lines
ALT 'I'		Insert separator
ALT BACK	UNDO	Undo one action in the undo history
ALT END	LINEENDDISPLAY	Move caret to last position on display line
ALT F4		Exit
ALT F9		Open terminal emulator
ALT HOME	HOMEDISPLAY	Move caret to first position on display line
ALT LEFT ARROW	WORDPARTLEFT	Move to the previous change in capitalisation
ALT RIGHT ARROW	WORDPARTRIGHT	Move to the next change in capitalisation
ALT+SHIFT END	LINEENDDISPLAYEXTEND	Move caret to last position on display line extending selection to new caret position
ALT+SHIFT HOME	HOMEDISPLAYEXTEND	Move caret to first position on display line extending selection to new caret position.
ALT+SHIFT LEFT ARROW	WORDPARTLEFTEXTEND	Move to the previous change in capitalisation extending selection to new caret position
ALT+SHIFT RIGHT ARROW	WORDPARTRIGHTEXTEND	Move to the next change in capitalisation extending selection to new caret position.
BACK	DELETEBACK	Dedent the selected lines
CTRL 'A'	SELECTALL	Select all the text in the document
CTRL 'B'		Load in Blender
CTRL 'C'	COPY	Copy the selection to the clipboard
CTRL 'D'	DEBUG	Debug
CTRL 'F'		Find & replace
CTRL 'G'		Go to line
CTRL 'K'		Test regular expression with Kiki
CTRL 'L'	LINECUT	Cut the line containing the caret
CTRL 'N'		New
CTRL 'O'		Open
CTRL 'P'		Run with profile
CTRL 'R'		Run in separate namespace
CTRL 'S'		Save
CTRL 'T'	LINETRANSPOSE	Switch the current line with the previous
CTRL 'U'	LOWERCASE	Transform the selection to lower case
CTRL 'V'	PASTE	Paste the contents of the clipboard into the document replacing the selection
CTRL 'X'	CUT	Cut the selection to the clipboard
CTRL 'Y'	REDO	Redoes the next action on the undo history
CTRL 'Z'	UNDO	Undo one action in the undo history
CTRL @		Contact author
CTRL ADD	ZOOMIN	Magnify the displayed text by increasing the sizes by 1 point
CTRL BACK	DELWORDLEFT	Delete the word to the left of the caret
CTRL DELETE	DELWORDRIGHT	Delete the word to the right of the caret
CTRL DIVIDE	SETZOOM	Set the zoom level to 0. This returns the zoom to 'normal,' i.e., no zoom.
CTRL DOWN ARROW	LINESCROLLDOWN	Scroll the document down, keeping the caret visible

Key	Action	Description
CTRL END	DOCUMENTEND	Move caret to last position in document
CTRL ENTER		Browse source
CTRL F4		Close
CTRL F9		Run in terminal emulator
CTRL HOME	DOCUMENTSTART	Move caret to first position in document
CTRL INSERT	COPY	Copy the selection to the clipboard
CTRL LEFT ARROW	WORDLEFT	Move caret left one word
CTRL RIGHT ARROW	WORDRIGHT	Move caret right one word
CTRL SPACE		Auto complete
CTRL SUBTRACT	ZOOMOUT	Make the displayed text smaller by decreasing the sizes by 1 point
CTRL UP ARROW	LINESCROLLUP	Scroll the document up, keeping the caret visible
CTRL+ALT 'B'		Reference in Blender
CTRL+ALT 'C'		Check source with PyChecker
CTRL+ALT 'F'		Browse Object with PyFilling
CTRL+ALT 'G'		Design a gui with wxGlade
CTRL+ALT 'P'		Preferences
CTRL+ALT 'R'		Run Verbose
CTRL+ALT 'X'		Design a gui with XRC
CTRL+ALT F9		Run in terminal emulator & exit
CTRL+SHIFT 'L'	LINEDELETE	Delete the line containing the caret
CTRL+SHIFT 'U'	UPPERCASE	Transform the selection to upper case
CTRL+SHIFT BACK	DELLINELEFT	Delete back from the current position to the start of the line
CTRL+SHIFT DELETE	DELLINERIGHT	Delete forwards from the current position to the end of the line
CTRL+SHIFT END	DOCUMENTENDEXTEND	Move caret to last position in document extending selection to new caret position
CTRL+SHIFT HOME	DOCUMENTSTARTEXTEND	Move caret to first position in document extending selection to new caret position
CTRL+SHIFT LEFT ARROW	WORDLEFTEXTEND	Move caret left one word extending selection to new caret position
CTRL+SHIFT RIGHT ARROW	WORDRIGHTEXTEND	Move caret right one word extending selection to new caret position
DELETE	CLEAR	Delete all text in the document
DOWN ARROW	LINEDOWN	Move caret down one line
END	LINEEND	Move caret to last position on line
ESCAPE	CANCEL	Cancel any modes such as call tip or auto-completion list display
F02		Save
F03		Find next
F05		Refresh
F09		Run
F10		Import
F11		Show/hide sidebar
F12		Show/hide shell
HOME	VCHOME	Move caret to before first visible character on line. If already there move to first character on line
INSERT	EDITTOGGLEOVERTYPE	Switch from insert to overtype mode or the reverse
LEFT ARROW	CHARLEFT	Move caret left one character
NEXT	PAGEDOWN	Move caret one page down
PRIOR	PAGEUP	Move caret one page up

Key	Action	Description
RETURN	NEWLINE	Insert a new line, may use a CRLF, CR or LF depending on EOL mode
RIGHT ARROW	CHARRIGHT	Move caret right one character
SHIFT BACK	BACKTAB	Delete the selection or if no selection, the character before the caret
SHIFT DELETE	CUT	Cut the selection to the clipboard
SHIFT DOWN ARROW	LINEDOWNEXTEND	Move caret down one line extending selection to new caret position
SHIFT END	LINEENDEXTEND	Move caret to last position on line extending selection to new caret position
SHIFT F9		Browse folder
SHIFT HOME	VCHOMEEXTEND	Like VCHome but extending selection to new caret position
SHIFT INSERT	PASTE	Paste the contents of the clipboard into the document replacing the selection
SHIFT LEFT ARROW	CHARLEFTEXTEND	Move caret left one character extending selection to new caret position
SHIFT NEXT	SCI_PAGEDOWNEXTEND	Move caret one page down extending selection to new caret position
SHIFT PRIOR	PAGEUPEXTEND	Move caret one page up extending selection to new caret position
SHIFT RETURN	NEWLINE	Insert a new line, may use a CRLF, CR or LF depending on EOL mode
SHIFT RIGHT ARROW	CHARRIGHTEXTEND	Move caret right one character extending selection to new caret position
SHIFT UP ARROW	LINEUPEXTEND	Move caret up one line extending selection to new caret position
TAB	TAB	If selection is empty or all on one line replace the selection with a tab character. If more than one line selected, indent the lines.
UP ARROW	LINEUP	Move caret up one line

13 Credits

Thanks to the following components SPE was made possible:

- Blender
    - 3D modeling, rendering, animation and game creation package
    - Copyright 2003 Blender Foundation - Ton Roosendaal
    - http://www.blender.org
- Kiki
    - free environment for regular expression testing (ferret)
    - Copyright 2003 Project 5 - Andrei
    - http://come.to/project5
- PyChecker
    - a python source code checking tool
    - Copyright (c) 2000-2001, MetaSlash Inc.
    - http://pychecker.sourceforge.net
- PyCrust
    - The flakiest python shell (Patrick K. O'Brien)
    - Sponsored by Orbtech - Your source for python programming expertise.
    - http://www.wxPython.org
- Pyframe guide to Wxpython
    - Documentation about wxStyledTextCtrl
    - Copyright 2003 Jeff Sasmor
    - http://www.pyframe.com/wxdocs/
- Pythonwin
    - python IDE and GUI Framework for Windows
    - Copyright 1994-2003 Mark Hammond
- Scintilla
    - Copyright 1998-2001 by Neil Hodgson
    - http://www.scintilla.org
- Sky icons
    - KDE icon theme made with gimp
    - Copyright 2002 David Vignoni
    - http://projects.dims.org/%7Edave/iconsky5.html
- WinPdb
    - A Remote Debugger for Python
    - Copyright 2005 Nir Aides
    - http://www.digitalpeers.com/pythondebugger

- wxGlade
    - wxGlade is a GUI designer written in Python with the popular GUI toolkit wxPython, that helps you create wxWindows/wxPython user interfaces. At the moment it can generate Python, C++ and XRC (wxWindows' XML resources) code.
    - Copyright 2003 Alberto Griggio, Marco Barisione, Marcello Semboli, Richard Lawson, D.H.
    - http://wxglade.sourceforge.net
- wxPython
    - python extension module for wxWindows GUI classes
    - Copyright 1997-2003 Robin Dunn and Total Control Software
    - http://www.wxPython.org

Special thanks to Tina Hirsch (Linux feedback).

Python is Copyright (c) 2000-2002 ActiveState Corp:

> Copyright (c) 2001, 2002 Python Software Foundation.
> All Rights Reserved.
>
> Copyright (c) 2000 BeOpen.com.
> All Rights Reserved.
>
> Copyright (c) 1995-2001 Corporation for National Research Initiatives.
> All Rights Reserved.
>
> Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
> All Rights Reserved.

This program uses IDLE extensions by Guido van Rossum, Tim Peters and others.

Alphabetical Index