Developing Python Applications for Maemo

Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

The article is believed to be still valid for the original topic scope.

Getting started with Python for Maemo 4.x

This tutorial explains how to develop Python applications for the Maemo 4.x. In order to understand this document, basic knowledge of the Maemo architecture is necessary. This tutorial is divided as follow:

- Python environment and how to setting it on Maemo 4.x
- Writing Python UI applications for Maemo 4.x

Python environment on Maemo 4.x

PyMaemo is the Python programming language support for the Maemo platform. It has been developed by INdT (Nokia Institute of Technology, Recife, Brazil) since 2004. PyMaemo was designed to be small and with a set of optimizations in order to achieve better performance when executed in Maemo-based devices. It makes available some important modules on the platform, such as BlueZ, GStreamer, Hildon and GTK+.

Therefore, besides C programming language, maemo platform programmers have support for another language to develop interesting applications. Python is a versatile and powerful option: it is used to develop sophisticated applications, such as Canola, as well as software prototypes.

PyMaemo has the following modules: iPython, PyBlueZ, PyGame, PyGconf, PyGnomeVFS, PyGobject, PyGtk, Pyid3lib, PyImage, Pyrex, Python, Python-dbus, PythonGPS-bt, Python-Gstreamer, Python-Hildon, Python-Numeric, Python-Osso, Python-Xml, Python-ABook, Evolution-python, Galago-python, PyC URL, Cython and Storm. Visit PyMaemo for more information about it.

Setting up PyMaemo environment

The first step is to setting up and testing the Maemo development environment. For more details, read http://maemo.org/development/documentation/tutorials/maemo_4-0_tutorial/ this] tutorial. After installing the Maemo development environment, there are two different targets: ARMEL, used for emulating applications for armel-based platforms, and X86 based platforms. For both cases Python development support is not installed by default. It is necessary to install PyMaemo and also the modules (in both the targets). To do so, execute the apt-get command to install python2.5-sdk package and all dependences, including modules.

```
[sbox-CHINOOK_X86: ~] > apt-get install python2.5-sdk
Reading package lists... Done
Building dependency tree... Done
The following extra packages will be installed:
libsdl-ttf2.0-0 python2.5 python2.5-bluez python2.5-bluez-dev python2.5-cairo python2.5-
cairo-dev python2.5-conic python2.5-conic-dev python2.5-dbus python2.5-dev
python2.5-gnome python2.5-gnome-dev python2.5-gobject python2.5-gobject-dev python2.5-
gstreamer python2.5-gstreamer-dev python2.5-gtk2 python2.5-gtk2-dev python2.5-hildon
python2.5-hildon-dev python2.5-id3lib python2.5-imaging python2.5-numeric python2.5-
numeric-dev python2.5-osso python2.5-osso-dev python2.5-pygame python2.5-runtime
python2.5-setuptools python2.5-xml python2.5-xml-dev
The following NEW packages will be installed:
libsdl-ttf2.0-0 python2.5-bluez python2.5-bluez-dev python2.5-cairo-dev python2.5-conic
python2.5-conic-dev python2.5-dbus python2.5-dev python2.5-gnome python2.5-gnome-dev
python2.5-gobject-dev python2.5-gstreamer python2.5-gstreamer-dev python2.5-gtk2-dev
python2.5-hildon-dev python2.5-id3lib python2.5-imaging python2.5-numeric-dev
```

```
python2.5-osso python2.5-osso-dev python2.5-pygame python2.5-runtime python2.5-sdk
python2.5-setuptools python2.5-xml python2.5-xml-dev
The following packages will be upgraded:
python2.5 python2.5-cairo python2.5-gobject python2.5-gtk2 python2.5-hildon python2.5-
numeric
6 upgraded, 26 newly installed, 0 to remove and 3 not upgraded.
Need to get 11.1MB/11.7MB of archives.
After unpacking 34.6MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
...
```

After successful installation of the Python development environment, we can start to implement some applications.

Writing Python UI applications for Maemo 4.x

This section explains how to use Python to develop applications for the Maemo 4.x platform. We start by developing a simple GTK+ application that creates and shows a window with the "Hello World" label. Observe that, an event to destroy and quit the program is sent whenever the window is closed.

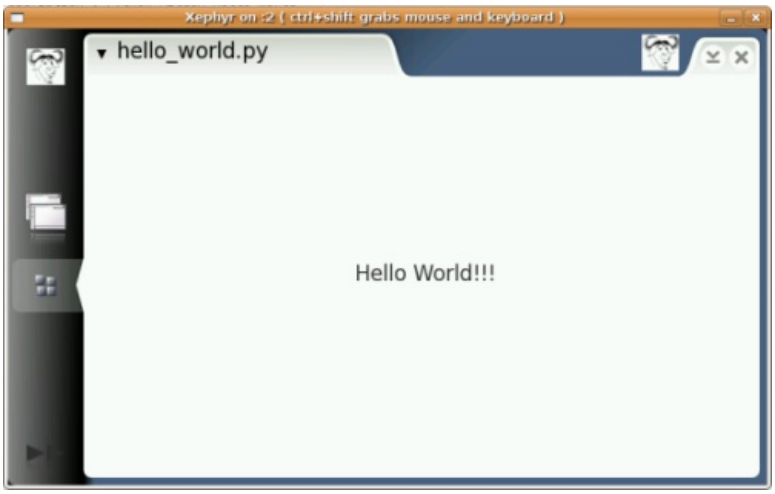
```
import gtk
if __name__ == "__main__":
window = gtk.Window(gtk.WINDOW_TOPLEVEL)

label = gtk.Label("Hello World!")
window.add(label)

label.show()
window.show()

gtk.main()
```

We create a top level GTK (gtk.Window(gtk.WINDOW_TOPLEVEL)) window that can contain other widgets. Then, we create a GTK label and we add it to the window. Finally, we show both the label and the window. The method gtk.main() is called to start main GUI loop. To execute the application, just type:

```
[sbox-CHINOOK_X86: ~] >python2.5 ./hello_world.py
```

The application was executed. However, it does not look like a Hildon theming application. For instance, the background is dark-gray and the fonts are small. It is necessary to execute the application using run-standalone.sh script in order to change to correct look-and-feel. Now, try again:

```
[sbox-CHINOOK_X86: ~] >run-standalone.sh python2.5 ./hello_world.py
```



But remember: run-standalone.sh script is only available if you run the application from the Scratchbox console or inside Internet Tablet.

Now, let's add a menu to our simple application. The function create_menu creates the menu and return it. We simply declare a gtk.Menu, append some gtk.MenuItems on it and finally add the menu to a gtk.MenuBar. Then, we create the box container to append the gtk.Label and the gtk.MenuBar.

```
import gtk

class HelloWorldApp:

 def __init__(self):
     self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)

     menu_bar = self.create_menu()
     label = gtk.Label("Hello World!")

     vbox = gtk.VBox(False, 0)
     vbox.pack_start(menu_bar, False, False, 0)
     vbox.pack_start(label, False, False, 0)

     self.window.add(vbox)

 def create_menu(self):
     menu = gtk.Menu()

     menuItemOpen = gtk.MenuItem("Open")
     menuItemSave = gtk.MenuItem("Save")
     menuItemSeparator = gtk.SeparatorMenuItem()
     menuItemExit = gtk.MenuItem("Exit")

     menu.append(menuItemOpen)
     menu.append(menuItemSave)
     menu.append(menuItemSeparator)
     menu.append(menuItemExit)

     menuItemFile = gtk.MenuItem("File")
     menuItemFile.set_submenu(menu)

     menu_bar = gtk.MenuBar()
     menu_bar.append(menuItemFile)
```

```
        return menu_bar

    def run(self):
        self.window.show_all()
        gtk.main()

if __name__ == "__main__":
    app = HelloWorldApp()
    app.run()
```

There is something wrong with menu position. Its correct location is in the title bar of the window. Remember that Hildon widgets are not being used, therefore some elements may be wrongly drawn. The platform contains a set of widgets that have been optimized for smaller devices with a stylus-like screen. In order to integrate nicely into the AF-environment, we change our GTK widgets to the one that Hildon expects us to use. Mainly, Hildon provides two widgets: Hildon Program and Hildon Window which replace some of the functionality that gtk.Window normally provides. Now, the menu is correctly drawn as expected in Hildon theming.

```
import gtk
import hildon

class HelloWorldApp(hildon.Program):

    def __init__(self):
        hildon.Program.__init__(self)

        self.window = hildon.Window()
        self.window.connect("delete_event", quit)
        self.add_window(self.window)

        menu_bar = self.create_menu()
        label = gtk.Label("Hello World!")

        self.window.set_menu(menu_bar)
        self.window.add(label)

    def quit(self, *args):
        gtk.main_quit()

    def create_menu(self):
        menu = gtk.Menu()

        menuItemOpen = gtk.MenuItem("Open")
```

```
        menuItemSave = gtk.MenuItem("Save")
        menuItemSeparator = gtk.SeparatorMenuItem()
        menuItemExit = gtk.MenuItem("Exit")

        menu.append(menuItemOpen)
        menu.append(menuItemSave)
        menu.append(menuItemSeparator)
        menu.append(menuItemExit)

        return menu

    def run(self):
        self.window.show_all()
        gtk.main()

if __name__ == "__main__":
    app = HelloWorldApp()
    app.run()
```

There are some small differences: we have to create both a hildon.Program and a hildon.Window. The hildon.Window class has a method to set its menu (set_menu(gtk.Menu)), so the vbox container is no more necessary.