by Hilaire Fernandes
<hilaire/at/ofset.org>

About the author:

Hilaire Fernandes is vice president of OFSET, an organization to promote and develop free software for education for the Gnome project. He has as well written Dr. Geo a software for dynamic geometry. Currently he is working on Dr. Genius an other mathematics software for educational purposes for the Gnome project.

Translated to English by:
Guido Socher
<guido/at/linuxfocus.org>

Developing Gnome applications with Python (part 2)

Abstract:

This article series is mainly intended for programming beginners in the area of Gnome and GNU/Linux. Python as the programming language has been selected because beginners are usually getting much faster into this language than compiled languages such as C. To understand this article you need some programming basics in Python.

Needed tools

The software needed to execute the described program was listed in the first article in this series.

You need as well:

- the file .glade, original [drill.glade] ;
- the drill Python source code [drill.py].

The installation procedure and the usage of Python-Gnome with LibGlade are as well described in the first part of this article series.

Drill, our support

The goal of the first article was to demonstrate the mechanism and the interaction modes between the different components of a program written for a configuration using Gnome, Glade, LibGlade and Python.

The example used the `GnomeCanvas` widget. This example provided us with a colorful presentation and showed us the ease of development with this configuration.

For the next sections, I suggest to work within a framework in which we will explain the different widgets of Gnome. This article concentrates on the setup of the framework. . Further articles will use this framework adding more features to illustrate the many Gnome widgets.

Our framework is called **Drill**. This is a platform for educational purposes which will be used for our examples and exercises. These examples are solely for educational purposes to demonstrate the usage of the widgets.

Creating an interface with Glade

The widgets

The application window is created with Glade. Like in the previous article you first create a window for a Gnome application. From that window you need to delete the useless icons and menus.

The main part of **Drill** has been divided into two workspaces using the `GtkPaned` widget.

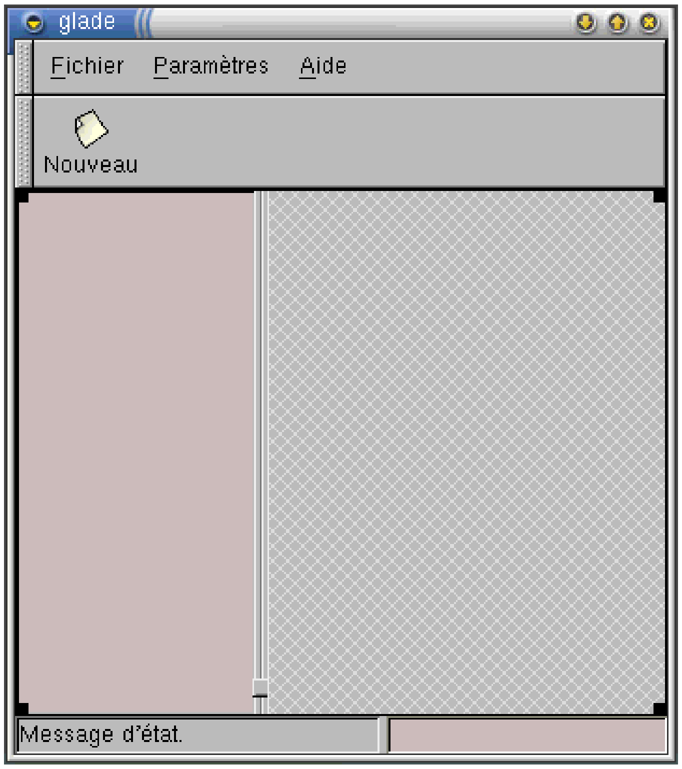


Fig. 1 - Drill main window

Both workspaces are vertically separated with a handle used to adjust the size of each. The left workspace contains the tree widget (`GtkTree`), in which the different parts of the exercise will be stored by category. The right workspace is empty. This is where we'll add the exercises according to the user's choice.

From Glade, viewing **Drill's** interface gives us insight into the structure of its components.

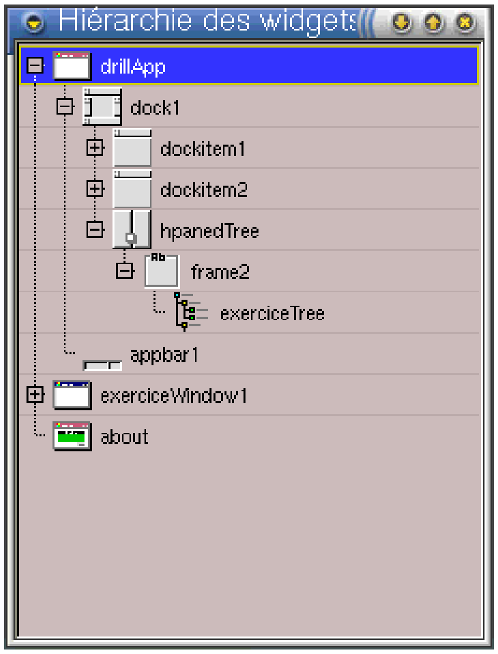
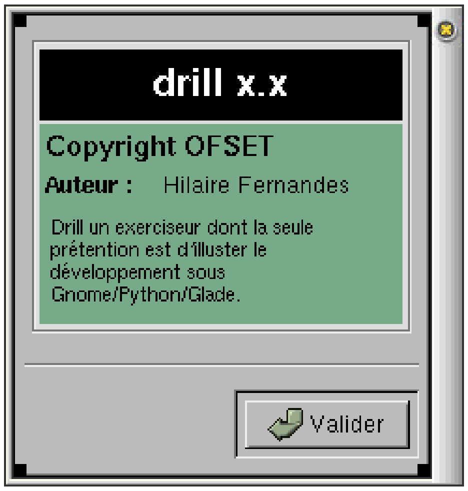


Fig. 2 - tree view of Drill

You can see in Fig. 2 that the `hpanedTree` widget (of the type `GtkPaned`) only contains one widget, `frame2` (of the type `GtkFrame`), the one on the left side. `frame2` contains the `exerciceTree` widget. It's preferable to first insert a `GtkFrame` widget with a shadow of the type `GTK_SHADOW_IN` into a `GtkPaned` widget. This avoids masking the handle.

Last, the Gnome dialog box "About **Drill**" can look like this one.



drill x.x

Copyright OFSET

Auteur : Hilaire Fernandes

Drill un exerciseur dont la seule prétention est d'illuster le développement sous Gnome/Python/Glade.

Fig. 3 - Dialog box "About" Drill

Its different items are edited from Glade in the `Widget` leaf from the `Properties` window.

The widgets and processing functions names

Use the following names for these widgets to manipulate them under these names from Python.

Gnome application window:
    `drillApp`
Handle separating the exercises tree :
    `hpanedTree`
Exercises tree:
    `exerciceTree`
Gnome dialog box About :
    `about`

You can see these widget names in Fig. 2

We list here quickly the names of the processing functions. If you need more information on this subject then you can read the part I of this article series.

Name of the widget	Signal	Processing
about	clicked	gtk_widget_destroy
about	close	gtk_widget_destroy
about	destroy	gtk_widget_destroy
button1 (Icon New in the tool bar)	clicked	on_new_activate
new	activate	on_new_activate
drillApp	destroy	on_exit_activate
exit	activate	on_exit_activate
about	activate	on_about_activate

Final adjustments

From Glade it is possible to specify the widgets geometry. In our case you can set the size of the `drillApp` to 400 and 300 from the `Common` tab in the `properties` panel. You can also set the horizontal divider position to 100 instead of 1.

Now the widget `exerciceTree` needs to be adjusted to only allow one selection at a time. As a matter of fact, only one exercise can be selected at a time. From the `properties` panel, select `Selection->Single`. The other options for this widget are less important.

Voilà! It’s all over as far as **Drill** is concerned. We’ll start developing exercises in the next article. For now, let’s see how to use the interface from Python and how to manipulate the `GtkTree` widget.

The Python code

The complete source code can be found at the end of this document. You need to save it in the same directory as the file `drill.glade`.

The required modules

```
from gtk import *
from gnome.ui import *
from GDK import *
from libglade import *
```

The graphical interface with LibGlade

The creation of the graphical interface and the connection of the processing functions with LibGlade is done in the same way as in the previous example. We won’t come back on this particular aspect.

In the Python program we define the global variables:

- `currentExercice`: Pointer to the widget that represents the current exercise. This last is placed in the right part of the **Drill** application window. The exercises will also be created from Glade.
- `exerciceTree` : Pointer to the tree widget on the left side of the **Drill** application window.
- `label` : Pointer to a label (`GtkLabel`). This label is a palliative to the fact that we don’t have any exercise at the moment. It will be placed on the right side of the tree -- where the exercises will be placed -- and we’ll display there the identifiers of the selected exercises.

The tree is created from LibGlade, its pointer is obtained via the following call:

```
exerciceTree = wTree.get_widget ("exerciceTree")
```

We need as well the pointer to the horizontal panels, in fact the container reference (`GtkPaned`) of the two horizontal panels separated by a handle. The one on the left contains the tree; the one on the right contains the exercises; for now we’ll place the label there :

```
  paned = wTree.get_widget ("hpanedTree")
  label = GtkLabel ("No exercise selected")
  label.show ()
  paned.pack2 (label)
```

Once again, the use of both the **GTK+ Reference manual** -- on the objects `GtkLabel` and `GtkPaned` -- and the source code of Python `/usr/lib/python1.5/site-packages/gtk.py` provide you with the required understanding on the right use of objects. .

The `GtkTree` widget

This is now the most important part of the article : how to use a tree of the `GtkTree` type.

The tree is filled up with consecutive calls to the `addMathExercices()`, `addFrenchExercices()`, `addHistoryExercices()` and `addGeographyExercices()` functions. These functions are all very similar. Each of these functions adds a new sub-category (a subtree) as well as titles of exercises (items) :

```
def addMathExercices ():
  subtree = addSubtree ("Mathematics")
```

```
   addExercice (subtree, "Exercise 1", "Math. Ex1")
   addExercice (subtree, "Exercise 2", "Math. Ex2")
```

The subtree

```
def addSubtree (name):
   global exerciceTree
   subTree = GtkTree ()
   item = GtkTreeItem (name)
   exerciceTree.append (item)
   item.set_subtree (subTree)
   item.show ()
   item.connect ("select", selectSubtree)
   return subTree
```

To create a subtree in an existing tree you need to do two things: Generate a `GtkTree` tree and a `GtkTreeItem` item, with the name of the subtree. Next, the item is added to the root tree -- the tree containing all categories -- and we add the subtree to the item using the `set_subtree()` method. Finally the `select` event is connected to the item, thus, when the category is selected, the `selectSubtree()` function is called.

GtkTreeItem

```
def addExercice (category, title, idValue):
   item = GtkTreeItem (title)
   item.set_data ("id", idValue)
   category.append (item)
   item.show ()
   item.connect ("select", selectTreeItem)
   item.connect ("deselect", deselectTreeItem)
```

The items have the names of the exercises as their title, here just `Exercice 1`, `Exercice 2`, ... To each item we associate an `id` additional attribute . GTK+ has the possibility to add to any object of the type `GtkObject` -- every GTK+ widgets comes from it -- some attributes. To do this there are 2 methods, `set_data (key, value)` and `get_data (key)` to initialize and get the value of an attribute. The item is then added to its category -- a subtree. Its `show()` method is called since it is required to force the display. Last, the `select` and `deselect` events are connected. The `deselect` event becomes active when the item looses its selection. Chronologically, the `deselectTreeItem()` method is called on the item loosing its selection, next `selectTreeItem()` is called on the item taking the selection.

The processing functions

We have defined three processing functions selectTreeItem(), `deselectTreeItem()` and `selectSubtree()`. They update the text label -- on the right side -- with the value of the `id` attribute. That’s all for now.

The final word

We just set the infrastructure in which we will add the exercises -- as many newly discovered widgets. We have mainly discussed the `GtkTree` widget and how to associate attributes to widgets. This mechanism is often used to get additional related information from the processing functions, what we have done here. Until the next article you can try to transform the **Couleur** game, we used in part I, as an exercise in **Drill**.

Appendix: The full source code

```
#!/usr/bin/python
# Drill - Teo Serie
# Copyright Hilaire Fernandes 2001
# Release under the terms of the GPL licence
# You can get a copy of the license at http://www.gnu.org from gtk import *
from gnome.ui import *
from GDK import *
from libglade import * exerciceTree = currentExercice = label = None

def on_about_activate(obj):
    "display the about dialog"
    about = GladeXML ("drill.glade", "about").get_widget ("about")
    about.show ()

def on_new_activate (obj):
    global exerciceTree, currentExercice

def selectTreeItem (item):
    global label
    label.set_text ("L’exercice " +
            item.get_data ("id") + "est sélectionné.")

def deselectTreeItem (item):
    global label
    label.set_text ("L’exercice " +
            item.get_data ("id") + "est désélectionné.")

def selectSubtree (subtree):
    global label
    label.set_text ("No selected exercise")

def addSubtree (name):
    global exerciceTree
    subTree = GtkTree ()
    item = GtkTreeItem (name)
    exerciceTree.append (item)
    item.set_subtree (subTree)
    item.show ()
    item.connect ("select", selectSubtree)
    return subTree

def addExercice (category, title, id):
    item = GtkTreeItem (title)
    item.set_data ("id", id)
    category.append (item)
    item.show ()
    item.connect ("select", selectTreeItem)
    item.connect ("deselect", deselectTreeItem)

def addMathExercices ():
    subtree = addSubtree ("Mathématiques")
    addExercice (subtree, "Exercice 1", "Math. Ex1")
    addExercice (subtree, "Exercice 2", "Math. Ex2")

def addFrenchExercices ():
    subtree = addSubtree ("Français")
    addExercice (subtree, "Exercice 1", "Français Ex1")
    addExercice (subtree, "Exercice 2", "Français Ex2")

def addHistoryExercices ():
    subtree = addSubtree ("Histoire")
    addExercice (subtree, "Exercice 1", "Histoire Ex1")
    addExercice (subtree, "Exercice 2", "Histoire Ex2")

def addGeographyExercices ():
    subtree = addSubtree ("Géographie")
    addExercice (subtree, "Exercice 1", "Géographie Ex1")
    addExercice (subtree, "Exercice 2", "Géographie Ex2")

def initDrill ():
    global exerciceTree, label
    wTree = GladeXML ("drill.glade", "drillApp")
    dic = {"on_about_activate": on_about_activate,
        "on_exit_activate": mainquit,
        "on_new_activate": on_new_activate}
    wTree.signal_autoconnect (dic)
    exerciceTree = wTree.get_widget ("exerciceTree")
    # Temporary until we implement real exercice
    paned = wTree.get_widget ("hpanedTree")
    label = GtkLabel ("No selected exercise")
    label.show ()
    paned.pack2 (label)
    # Free the GladeXML tree
    wTree.destroy ()
    # Add the exercices
    addMathExercices ()
    addFrenchExercices ()
    addHistoryExercices ()
    addGeographyExercices ()

initDrill ()
mainloop ()
```
