LinuxFocus article number 224
http://linuxfocus.org

Développer des Applications Gnome avec Python (Partie 2)

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

L´auteur:

Hilaire Fernandes est le vice-président d'OFSET, une organisation pour promouvoir le développement de logiciels éducatifs libres pour le bureau Gnome. Il a aussi écrit Dr.Geo, un logiciel primé de géométrie dynamique, et il est actuellement occupé avec Dr.Genius un autre logiciel éducatif de mathématiques pour le bureau Gnome.

Résumé:

Cette série d'articles est spécialement écrite pour des débutants en programmation sous Gnome et GNU/Linux. Le langage de développement choisi, Python, évite la surcharge habituelle avec des langages compilés comme le C. Avant d'étudier cet article quelques notions de programmation sous Python sont nécessaires.

Outils nécessaires

Pour les besoins logiciels à l'exécution du programme décrit dans cet article, vous pouvez vous référer à la liste de la même rubrique de la partie I de cette série d'articles.

Vous aurez aussi besoin :

- du fichier .glade original [drill.glade] ;
- du code source en Python [drill.py].

Pour l'installation et l'utilisation de Pyhton-Gnome et LibGlade vous pouvez aussi vous référer à la partie I.

Drill, notre support

La première partie avait pour objectif de montrer les mécanismes et les modes d'interactions entre les différents composants d'un programme écrit sous une configuration de type Gnome, Glade, LibGlade et Python.

L'exemple utilisait le widget `GnomeCanvas`. Celui-ci nous a offert une illustration riche en couleur montrant l'intérêt et la facilité de développement sous cette configuration.

Pour les parties suivantes, je vous propose de mettre en place un cadre logiciel dans lequel nous illustrerons les différents widgets de Gnome. Le présent article s'attache principalement à mettre en place ce cadre. Les articles suivants s'appuieront sur celui-ci, en lui ajoutant des fonctionnalités, pour illustrer les différents widgets de Gnome.

Notre cadre logiciel s'appelle **Drill**. C'est une plate-forme à caractère éducatif sur laquelle nous grefferons des exercices. Attention, les exercices ont pour seule prétention pédagogique d'illustrer l'usage des widgets !

Construction de l'interface avec Glade

Les widgets

La fenêtre de l'application est créée à l'aide de Glade. Comme dans l'article précédent, vous créez dans un premier temps une fenêtre d'une application Gnome. Dans celle-ci vous supprimez les menus et icônes inutiles.

La zone principale de **Drill** est subdivisée en deux espaces grâce au widget `GtkPaned`.

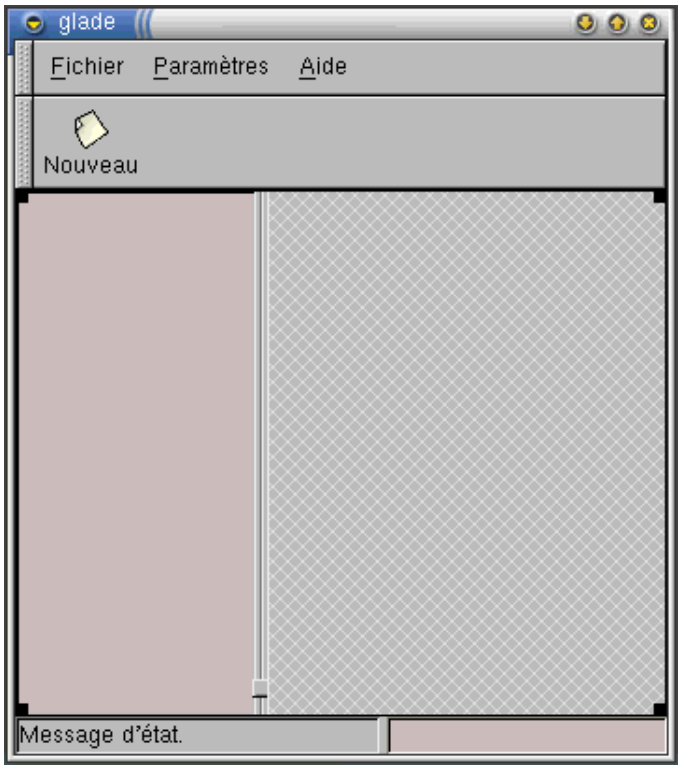

Fig. 1 - Fenêtre principale de Drill

Ils sont séparés verticalement par une poignée permettant d'ajuster la subdivision. L'espace de gauche est occupé par un arbre (widget `GtkTree`) dans lequel seront rangés par catégorie les intitulés des exercices. L'espace à droite est vide, c'est ici que nous grefferons les exercices eux-mêmes en fonction du choix de l'utilisateur.

Depuis Glade, la vue de l'interface de **Drill** sous forme d'arbre permet de comprendre son agencement :

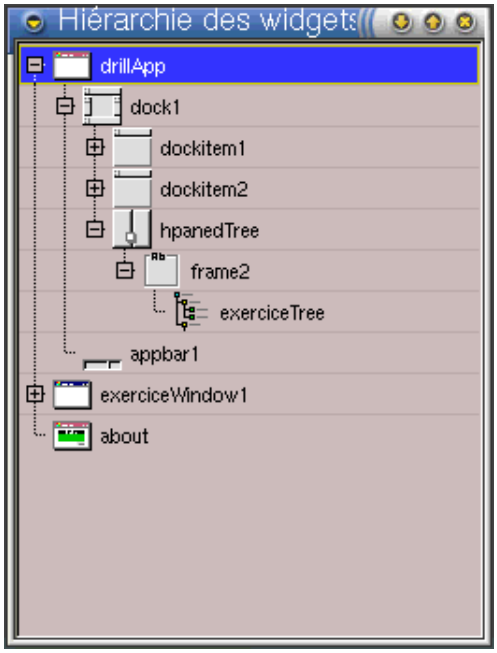

Fig. 2 - Vue en arbre de l'interface de Drill

Dans la Fig. 2, nous voyons que le widget nommé `hpanedTree` (de type `GtkPaned`) ne contient qu'un seul widget, `frame2` (de type `GtkFrame`), c'est celui-ci qui est à gauche. `frame2` contient lui-même le widget `exerciceTree`. En effet, il est préférable de placer d'abord un widget `GtkFrame` avec une ombre de type `GTK_SHADOW_IN` dans un widget `GtkPaned`, cela évite de mordre sur la poignée.

Pour finir la boîte de dialogue Gnome "À propos" de **Drill** peut ressembler à celle-ci

Fig. 3 - Boîte de dialogue "À propos" de Drill

Ses différentes rubriques sont éditées depuis Glade, dans le feuillet `Widget` de la fenêtre `Propriétés`.

Les noms des widgets et des fonctions de traitement

Appliquez les noms suivants à ces widgets afin de les manipuler sous ces noms depuis Pyhton.

Fenêtre d'application Gnome :
`drillApp`
Poignée séparant l'arbre des exercices :
`hpanedTree`
Arbre des exercices :
`exerciceTree`
Boîte de dialogue Gnome À-propos :
`about`

Ces widgets sont ceux dont les noms sont visibles sur la Fig. 2

Nous listons ici rapidement les noms des fonctions de traitement. Si vous avez besoin d'informations complémentaires sur le sujet référez-vous à la partie I.

Nom de widget	Signal	Traitement
about	clicked	gtk_widget_destroy
about	close	gtk_widget_destroy
about	destroy	gtk_widget_destroy
button1 (icône nouveau dans la barre à outils	clicked	on_new_activate
new	activate	on_new_activate
drillApp	destroy	on_exit_activate
exit	activate	on_exit_activate
about	activate	on_about_activate

Derniers ajustements

Depuis Glade il est possible de spécifier la géométrie des widgets. Dans notre affaire, vous pouvez ajuster la taille de `drillApp` à 400 et 300 depuis l'onglet `Commun` du panneau de `Propriétés`. Aussi, la position du diviseur des panneaux horizontaux peut être ajustée à 100 au lieu de 1.

Ensuite, le widget `exerciceTree` doit être ajusté afin de ne permettre qu'une seule sélection à la fois. En effet un seul exercice peut être sélectionné à la fois. Depuis le panneau de `Propriétés`, choisir `Selection->Single`. Les autres options de ce widget sont de moindre importance.

Voilà! C'est fini en ce qui concerne **Drill** lui-même. Nous commencerons à développer des exercices dès le prochain article. Pour le moment nous allons voir comment utiliser l'interface depuis Python et nous intéresser à la manipulation du widget `GtkTree`.

Le code Python

Le code source complet se trouve à la fin de ce document. Il doit être sauvegardé dans le même dossier que le fichier `drill.glade`.

Les modules nécessaires

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

L’interface graphique avec LibGlade

La construction de l’interface graphique et la connection des fonctions de traitement avec LibGlade se fait de façon analogue à l’exemple précédent. Nous ne revenons pas sur cet aspect là.

Dans le programme python nous définissons des variables globales :

- `currentExercice`: référence du widget représentant l’exercice courant. Celui-ci est placé dans la partie droite de la fenêtre d’application de **Drill**. Les exercices seront également créés à partir de Glade.
- `exerciceTree` : référence de l’arbre à gauche dans la fenêtre d’application de **Drill**.
- `label` : référence un label (`GtkLabel`). Ce label est un palliatif à l’absence d’exercice pour le moment. Il sera donc placé à droite de l’arbre -- où les exercices prendront place -- et nous y afficherons les identifiants des exercices sélectionnés.

L’arbre est créé par LibGlade, sa référence est récupérée par l’appel suivant:

exerciceTree = wTree.get_widget ("exerciceTree")

Nous avons également besoin de la référence des panneaux horizontaux, en fait la référence du conteneur (`GtkPaned`) des deux panneaux horizontaux séparés par une poignée. Celui à gauche contient l’arbre ; celui à droite les exercices, nous y placerons pour le moment le label :

paned = wTree.get_widget ("hpanedTree") label = GtkLabel ("Aucun exercice de sélectionné")
label.show () paned.pack2 (label)

Une fois encore l’utilisation conjointe du **Manuel de référence de GTK+** -- sur les objets `GtkLabel` et `GtkPaned` -- et du source Python `/usr/lib/python1.5/site-packages/gtk.py` offre le discernement nécessaire à la bonne utilisation des objets.

Le widget `GtkTree`

Nous arrivons ici à l’élément essentiel de notre article, à savoir l’utilisation d’un arbre de type `GtkTree`.

L’arbre est rempli par les appels successifs aux fonctions `addMathExercices()`, `addFrenchExercices()`, `addHistoryExercices()` et `addGeographyExercices()`. Elles sont toutes semblables. Chacune de ces fonctions ajoutent une sous catégorie (un sous arbre) ainsi que des titres d’exercices (les items):

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

addExercice (subtree, "Exercice 2", "Math. Ex2")

Le sous-arbre

```
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
```

Pour créer un sous-arbre dans un arbre existant, il faut créer deux choses : un arbre `GtkTree` et un item `GtkTreeItem` portant le nom du sous-arbre. Ensuite l'item est ajouté à l'arbre racine -- notre arbre contenant toutes les catégories -- puis nous greffons notre sous-arbre à l'item grâce à sa méthode `set_subtree()`. Enfin, l'événement `select` est connecté à l'item, ainsi lorsque la catégorie est sélectionnée, la fonction `selectSubtree()` est appelée.

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)
```

Les items portent comme titre les noms des exercices, ici en général simplement `Exercice 1`, `2`, ... À chaque item nous associons un attribut supplémentaire, `id`. GTK+ offre en effet la possibilité d'ajouter à tout objet de type `GtkObject` -- dont tous les widgets de GTK+ sont issus -- des attributs. Pour ce faire il existe deux méthodes `set_data (key, value)` et `get_data (key)` pour initialiser et récupérer la valeur d'un attribut. L'item est ensuite ajouté à sa catégorie -- un sous arbre. Sa méthode `show()` est appelée, elle est nécessaire pour forcer l'affichage. Enfin les événements `select` et `deselect` sont connectés, l'événement `deselect` prend lieu lorsque l'item perd la sélection. Chronologiquement, la méthode `deselectTreeItem()` est appelée sur l'item perdant la sélection, puis `selectTreeItem()` est invoquée sur l'item prenant la sélection.

Les fonctions de traitement

Nous avons défini trois fonctions de traitement `selectTreeItem()`, `deselectTreeItem()` et `selectSubtree()`. Ces méthodes mettent à jour le texte du label -- placé dans la zone de droite -- avec la valeur de l'attribut `id`. C'est tout pour le moment.

Le mot final

Nous avons ici mis en place l'infrastructure dans laquelle nous grefferons des exercices -- autant de nouveaux widgets que nous découvrirons. Nous avons principalement étudié le widget `GtkTree` et comment associer des attributs à des widgets. Ce dernier mécanisme est très souvent utilisé pour récupérer dans les fonctions de traitement des informations supplémentaires associées, ce que nous avons fait ici. En attendant le prochain article, vous pouvez essayer de transformer le jeu **Couleur**, étudié en partie 1, comme un exercice dans **Drill**.

Appendice: Le source complet

```
#!/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 ("Aucun exercice de sélectionné")

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 ("Aucun exercice de sélectionné")
label.show ()
paned.pack2 (label)
# Free the GladeXML tree
wTree.destroy ()
# Add the exercices
addMathExercices ()
addFrenchExercices ()
addHistoryExercices ()
addGeographyExercices ()

initDrill ()
mainloop ()
```

2005-01-14, generated by lfparser_pdf version 2.51