LinuxFocus article number 224
http://linuxfocus.org

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

Over de auteur:

Hilaire Fernandes is vice president van OFSET, een organisatie om educatieve free software voor het Gnome project te promoten en te ontwikkelen. Hij is ook de auteur van Dr. Geo, een programma voor dynamische geometrie. Op het moment werkt hij aan Dr. Genius, een ander wiskundig programma voor onderwijsdoeleinden voor het Gnome project.

Gnome applicaties ontwikkelen met Python (deel 2)

Kort:

Deze serie artikelen is vooral bedoeld voor beginnende programmeurs op het gebied van Gnome en GNU/Linux. Python is geselecteerd als programmeertaal omdat beginners hier vaak sneller mee leren werken dan met gecompileerde talen als C. Om dit artikel te begrijpen is enig inzicht in de programmeer basics van Python nodig.

Vertaald naar het Nederlands door:
Guus Snijders
<ghs(at)linuxfocus.org>

Benodigdheden

De software die nodig is om het beschreven programma uit te voeren werd genoemd in het eerste artikel in deze serie.

Verder heb je nodig:

- het .glade bestand, origineel [drill.glade] ;
- de Drill Python bron code [drill.py].

De installatie procedure en het gebruik van Python-Gnome met LibGlade zijn ook beschreven in het eerste deel van deze artikel serie.

Drill, onze ondersteuning

Het doel van het eerste artikel was om de mechanismen en de interactie modi tussen de verschillende componenten van een programma te demonstreren, geschreven voor een configuratie met Gnome, Glade, LibGlade en Python.

Het gebruikte voorbeeld maakte gebruik van de `GnomeCanvas` widget. Dit voorbeeld leverde ons een kleurige presentatie en gaf de eenvoud van ontwikkeling van deze configuratie weer.

Voor de volgende secties, stel ik voor te werken in een framework waarin we de verschillende widgets voor Gnome uitleggen. Dit artikel concentreert op het opzetten van dit framework. . Verdere artikelen zullen dit framework gebruiken en meer features toevoegen om de vele Gnome widgets te demonstreren.

Ons framework draagt de naam **Drill**. Dit is een platform voor educatieve doelen, en zal gebruikt worden voor onze voorbeelden en oefeningen. Deze voorbeelden zijn slechts voor leer doeleinden om het gebruik van de widgets te laten zien.

Een interface bouwen met Glade

De widgets

Het applicatie venster is gemaakt met Glade. Net als in het eerste artikel, maak je eerst een venster voor een Gnome applicatie. Vanuit dat window verwijder je de overbodige pictogrammen en menus.

Het belangrijkste deel van **Drill** is verdeeld in twee workspaces met behulp van de `GtkPaned` widget.

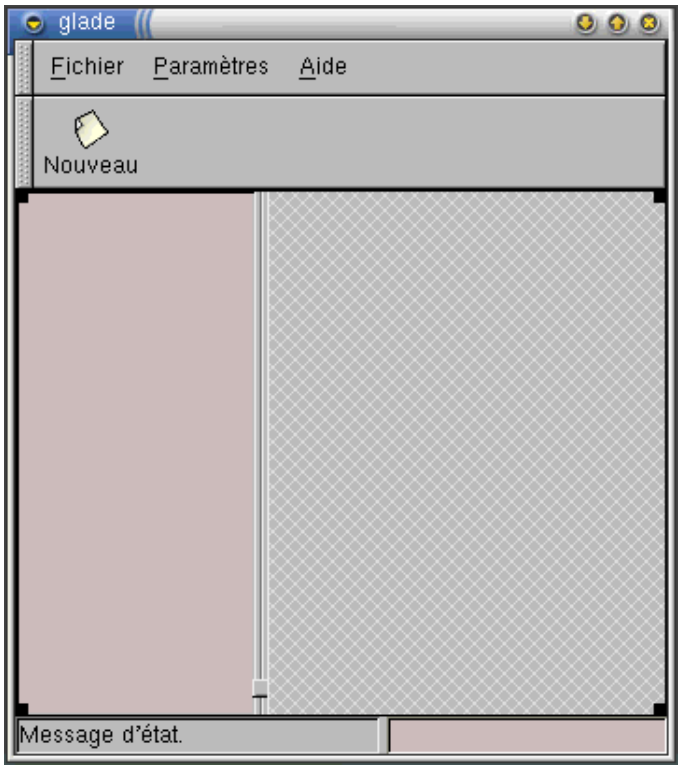

Fig. 1 - Drill main venster

Beide workspaces zijn verticaal gescheiden met een handle welke ook gebruikt kan worden om de grootte van de twee aan te passen. De linker workspace bevat de tree (boom) widget (`GtkTree`), waarin de verschillende onderdelen van de oefening worden opgeslagen per categorie. De rechter workspace is leeg. Dit is waar we de oefeningen zullen toevoegen, aan de hand van de keuze van de gebruiker.

Het bekijken van de interface van **Drill** levert ons inzicht in de structuur van zijn componenten.

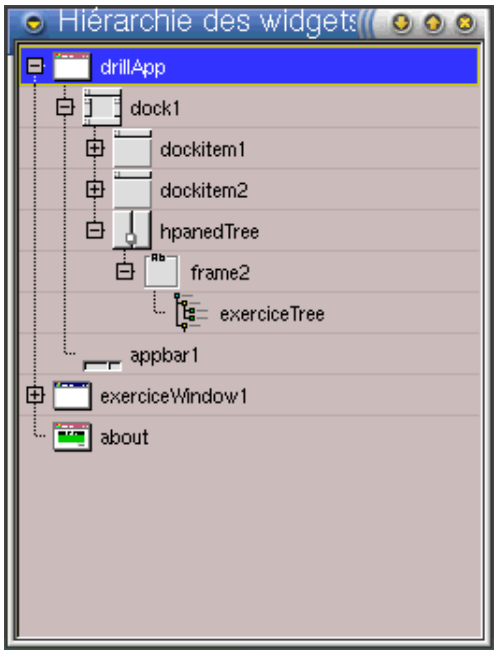

Fig. 2 - boom weergave van Drill

In figuur 2 kun je zien dat de `hpanedTree` widget (type `GtkPaned`) slechts een widget bevat, `frame2` (type `GtkFrame`), degene aan de linkerkant. `frame2` bevat de `execiseTree` widget. Het is aan te bevelen om eerst een `GtkFrame` widget met schaduw van het type `GTK_SHADOW` widget toe te voegen in een `GtkPaned` widget. Dit voorkomt het maskeren van de handle.

Ten slotte, de Gnome "About **Drill**" dialoog box kan er als volgt uitzien.

Fig. 3 - "About" Drill Dialoog box

De overige items worden bewerkt in Glade in het `Widget` blad van het `Properties` venster.

De widget en proces functie namen

Gebruik de volgende namen voor deze widgets om ze te bewerken onder deze namen vanuit Python.

Gnome applicatie venster:
`drillApp`
Handle voor het scheiden van de boom:
`hpanedTree`
Oefeningen boom:
`exerciceTree`
Gnome dialoog box About:
`about`

Deze widget namen zijn te zien in Fig. 2

Hier een korte lijst van de proces functies. Als je meer informatie nodig hebt over dit onderwerp, kun je deel I van deze serie lezen.

Naam van de widget	Signaal	Bewerking
about	geklikt	gtk_widget_destroy
about	sluiten	gtk_widget_destroy
about	vernietig	gtk_widget_destroy
button1 (Icoon New in de werkbar)	geklikt	on_new_activate
new	activeer	on_new_activate
drillApp	vernietig	on_exit_activate
exit	activeer	on_exit_activate
about	activeer	on_about_activate

Laatste aanpassingen

Vanuit Glade is het mogelijk om de afmetingen van de widgets op te geven. In ons geval kun je de grootte van `drillApp` op 400 en 300 zetten, vanuit de `Common` tab in het `properties` paneel. Je kunt ook de horizontale scheider posititie op 100 zetten in plaats van 1.

Nu moet de widget `exerciceTree` aangepast worden zodat deze een selectie tegelijk toestaat. In feite kan slechts een oefening tegelijk worden geselecteerd. Vanuit het `properties` paneel, selecteer `Selection->Single`. De andere opties voor deze widget zijn minder belangrijk.

Voilà! Klaar, voor zover het **Drill** aangaat. We zullen beginnen met het ontwikkelen van oefeningen in het volgende artikelen. Laten we voor nu kijken hoe we de interface vanuit Python kunnen gebruiken en hoe we de `GtkTree` widget kunnen manipuleren.

De Python code

De complete broncode kan aan het eind van dit artikel gevonden worden. Deze wordt opgeslagen in dezelfde directory als het bestand `drill.glade`.

De benodigde modules

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

from GDK import *
from libglade import *

De graphische interface met LibGlade

Het bouwen van de grafische interface en de connectie van de verwerkings functies met LibGlade gebeurt op dezelfde manier als in het vorige voorbeeld. We zullen niet terugkeren op dit particuliere aspect.

In het python programma definiëren we de globale variabelen:

- `currentExercice` : Pointer naar het widget dat de huidige oefening representeerd. Deze laaste is geplaatst in het rechterdeel van het **Drill** applicatie venster. De oefeningen zullen ook gemaakt worden vanuit Glade.
- `exerciceTree` : Pointer naar het tree widget aan de linkerkant van het **Drill** applicatie venster.
- `label` : Pointer naar een label (`GtkLabel`). Dit label is een lapmiddel voor het feit dat we op het moment geen oefening hebben. Deze zal aan de rechterkant van de boom worden geplaatst -- waar de oefeningen links zullen worden geplaatst -- en we zullen het hier de identifiers van de geselecteerde oefeningen weergeven.

De boom is gebouwd met LibGlade, de pointer wordt verkregen met de volgende call:

exerciceTree = wTree.get_widget ("exerciceTree")

We hebben ook de pointer naar de horizontale panelen nodig, in feite de container referentie (`GtkPaned`) van de twee horizontale panelen, gescheiden door een handle. De linker bevat de tree; de rechter bevat de oefeningen; voor nu zullen we het label daar plaatsen:

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

Nogmaals, het gebruik van zowel de **GTK+ Reference manual** -- over de objecten `GtkLabel` en `GtkPaned` -- en de broncode van Python `/usr/lib/python1.5/site-packages/gtk.py` leveren je het benodigde begrip van het juiste gebruik van objecten.

Het `GtkTree` widget

Dit is het belangrijkste deel van het artikel: het gebruik van een boom van het `GtkTree` type.

De boom is gevuld met opeenvolgende calls naar de `addMathExercices()`, `addFrenchExercices()`, `addHistoryExercices()` en `addGeographyExercices()` functies. Deze functies zijn alle erg verglijkbaar. Ieder van deze functies voegt een nieuwe sub-category (een subtree of vertakking) alsook

de titels van oefeningen (items) :

```
def addMathExercices ():
  subtree = addSubtree ("Mathematics")
  addExercice (subtree, "Exercise 1", "Math. Ex1")
  addExercice (subtree, "Exercise 2", "Math. Ex2")
```

De 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
```

Om een subtree (vertakking) te maken in een bestaande boom, doe je twee dingen: Genereer een `GtkTree` boom en een `GtkTreeItem` item, met de naam van de vertakking. Vervolgens wordt het item toegevoegd aan de root tree -- de boom die alle categorieën bevat -- en we voegen de subtree toe aan het item met de `set_subtree()` methode. Tenslotte wordt de `select` gebeurtenis verbonden met het item, dus, als de category wordt geselecteerd, wordt de `selectSubtree()` functie aangeroepen.

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)
```

De items hebben de namen van de oefeningen als titel, hier zijn dat `Exercice 1`, `Exercice 2`, ... Voor ieder item associëren we een additioneel `id` attribuut. GTK+ heeft de mogelijkheid om een aantal attributen toe te kennen aan objecten van het type `GtkObject` -- iedere GTK+ widget komt hier vandaan --. Er zijn twee methoden om dit te doen, `set_data (sleutel, waarde)` en `get_data (sleutel)` om te initialiseren en de waarde van een attribuut te krijgen. Het item wordt dan toegevoegd aan zijn categorie -- een subtree. Zijn `show()` methode wordt aangeroepen daar het vereist is om de weergave te forceren. Tenslotte, de `select` en `deselect` gebeurtenissen zijn verbonden. De `deselect` gebeurtenis wordt actief als het item de selectie verliest. Chronologisch wordt de `deselectTreeItem()` methode aangeroepen op het item dat zijn selectie verliest, vervolgens wordt `selectTreeItem()` wordt aangeroepen op het Item dat de selectie krijgt.

De verwerkings functies

We hebben drie verwerkingsfuncties gedefinieerd, namelijk selectTreeItem(), `deselectTreeItem()` en `selectSubtree()`. Deze updaten het tekst label -- aan de rechterkant -- met de waarde van het `id` attribuut. Dat is alles voor nu.

Het laatste woord

We hebben zojuist de infrastructuur opgezet waarin we de oefeningen zullen toevoegen -- zoals vele nieuw-gevonden widgets. We hebben voornamelijk de `GtkTree` widget besproken en hoe attributen te associëren met widgets. Dit mechanisme wordt vaak gebruikt om aanvullende, gerelateerde informatie over de verwerkingsfuncties te krijgen, wat we hier ook hebben gedaan. Tot het volgende artikel verschijnt, kun je proberen het **Couleur** spel, gebruikt in deel I, te transformeren, als een oefening in **Drill**.

Appendix: De volledige broncode

```
#!/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 ()
```

2005-01-14, generated by lfparser_pdf version 2.51