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

About the author:

Hilaire Fernandes ist Vizepräsident OFSET, einer Gesellschaft zur Förderung und Entwicklung von freier Schul– und Lernsoftware für das Gnome–Projekt. Er hat ebenfalls Dr. Geo, eine preisgekrönte Software für dynamische Geometrien geschrieben. Zur Zeit beschäftigt er sich mit Dr. Genius, einer anderen mathematischen Lernsoftware für das Gnome Projekt.

Entwickeln von Gnome Applikationen mit Python (Teil 2)

Abstract:

Diese Artikelreihe ist vor allem für Anfänger in der Programmierung mit Gnome und GNU/Linux geschrieben. Die gewählte Programmiersprache Python ist einfach und vermeidet die gewöhnliche Überforderung des Benutzers wie z. B. mit der Compilersprache C. Um diesen Artikel zu verstehen, sind einige Grundkenntnisse in der Programierung mit Python notwendig.

Was man braucht

Eine Liste der notwendigen Software zur Ausführung des beschriebenen Programmes befindet sich im Teil I dieser Artikelreihe.

Ebenfalls benötigen Sie :

- die Datei .glade original [drill.glade] ;
- den Sourcecode von Python [drill.py].

Die Installation und die Anwendung von Python–Gnome und LibGlade ist ebenfalls im Teil I beschrieben.

Drill, unser Support

Ziel des ersten Teils war es, die verschiedenen Mechanismen und das Zusammenspiel der verschiedenen Teile eines unter Gnome–, Glade–, LibGlade– und Python– geschriebenen Programmes zu zeigen.

Das Anwendungsbeispiel benutzt das Widget `GnomeCanvas`. Das ermöglicht uns eine farbenreiche Darstellung und zeigt die Vorteile und die Einfachheit der Programmentwicklung mit dieser Konfiguration.

Im folgenden Abschnitt werden die verschiedenen Widgets von Gnome dargestellt. Dieser Artikel beschäftigt sich hauptsächlich damit, diese Rahmenbedingungen zu erklären. Weitere Artikel, die sich auf diesen beziehen werden, fügen noch mehr Funktionen hinzu, um die verschiedenen Widgets (=Elemente in einer grafischen Oberfläche, z.B. ein Knopf) von Gnome zu erklären

Unsere wichtigste Software nennt sich ***Drill***. Es handelt sich um eine Platform zur Schulung, auf die sich unsere Beispiele und Übungen beziehen. Diese Beispiele und Übungen dienen ausschließlich zu pädagogischen Zwecken, um den Umgang mit den Widgets zu zeigen.

Erstellung der Schnittstelle mit Glade

Die Widgets

Das Anwendungsfenster wird mit Hilfe von Glade erstellt. Wie im vorherigen Artikel erstellen Sie zunächst ein Fenster für eine Gnome Applikation. Darin müssen Sie die Icons und Menüs löschen.

Der Haupteil von ***Drill*** ist mit dem Widget `GtkPaned` in zwei Teile unterteilt worden.

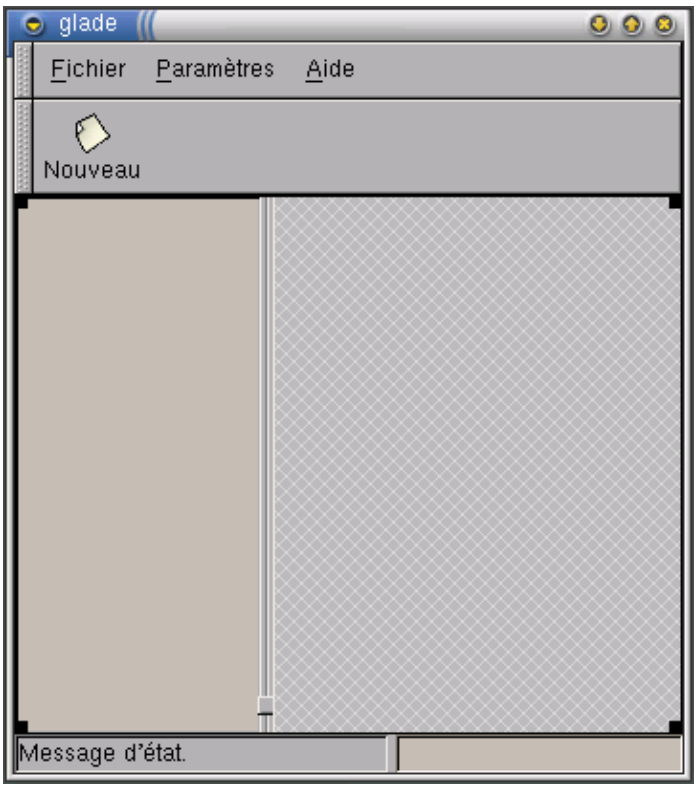

Abb. 1 – Fenster von Drill

Sie sind vertikal unterteilt und diese Unterteilung lässt sich auch verschieben. Der linke Teil enthält ein Baum Widget (`GtkTree`), in welchem die verschiedenen Abschnitte der Übung angeordnet sind. Der rechte Teil ist leer und dort werden die Übungen von dem Anwender dann ausgeführt.

Glade kann das Widget Interface von ***Drill*** in Form eines hirarchischen Baumes darstellen. Das erleichtert das Verständnis.

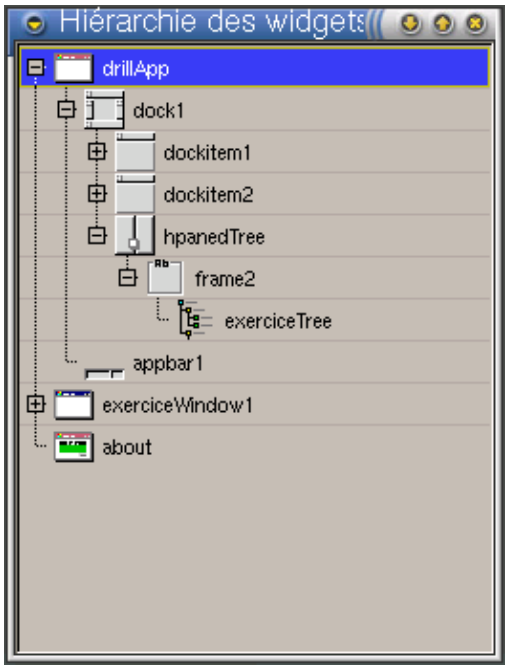

Abb. 2 – Baumdarstellung der Schnittstelle von Drill

In der Abb. 2 kann man das Widget `hpanedTree` sehen (vom Typ `GtkPaned`). Es enthält nur einen einzigen Widget, `frame2` (vom Typ `GtkFrame`), dieser befindet sich links. `frame2` enthält den widget `exerciceTree`. Es ist vorteilhafter zunächst ein Widget `GtkFrame` mit der Schattierung des Typs `GTK_SHADOW_IN` in einem `GtkPaned` Widget, zu nehmen, was Überschneidungen mit der Trennleiste verhindert.

Zum Schluß das Dialogfenster Gnome "À propos", das beim Beenden von ***Drill*** erscheint.

Abb. 3 – Dialogfenster "À propos" von Drill

Die unterschiedlichen Elemente sind von Glade aus im Dialogfenster `Widget` des Fensters `Eigenschaften` bearbeitet worden.

Die Namen der Widgets und seine Bedingungsfunktionen

Benutzen Sie die folgenden Namen für diese Widgets, um diese schließlich unter diesen Namen unter Python zu verändern.

Anwendungsfenster Gnome :
`drillApp`
Die Trennleiste zwischen dem Baum und der Übung :
`hpanedTree`
Baum der Übungen :
`exerciceTree`
Dialogfenster Gnome À–propos :
`about`

Diese widgets sind sichtbar auf der Abb. 2 bezeichnet

Wir führen hier schnell die Bedienungsfunktionen auf. Falls Sie weitergehende Informationen zu diesem Thema benötigen, können Sie diese im Teil I dieser Artikelserie nachschlagen

Name des Widgets	Signal	Event handler
about	clicked	gtk_widget_destroy
about	close	gtk_widget_destroy
about	destroy	gtk_widget_destroy
button1 (Neues Icon in der Arbeistleiste)	clicked	on_new_activate
new	activate	on_new_activate
drillApp	destroy	on_exit_activate
exit	activate	on_exit_activate
about	activate	on_about_activate

Letzte Anpassungen

Von Glade aus ist es möglich, die Geometrie der Widgets zu definieren. In unserem Fall können Sie die Größe der `drillApp` auf 400x300 im Eigenschaftendialogfenster einstellen. Ebenfalls kann die Position der horizontalen Unterteilung des Fensters auf 100 statt 1 eingestellt werden.

Anschließend muss das Widget `exerciceTree` eingestellt werden, um nur eine Auswahl gleichzeitig zu erlauben. Nur eine Übung gleichzeitig ausgewählt werden. Von dem Fenster `Eigenschaften`, wählt man `Selection->Single`. Die anderen Optionen von diesem Widget sind weniger wichtig

Voilà! Was ***Drill*** betrifft, ist die Konfiguration fertig. Jetzt fangen wir an, die Übungen des Artikels auszuführen. Nun werden wir sehen, wie man die Schnittstelle von Python aus benutzt und wir werden uns

mit den Einstellungen des Widgets `GtkTree` beschäftigen.

Der Code von Python

Der vollständige Sourcecode befindet sich am Ende dieses Dokumentes. Er muss im gleichen Verzeichnis wie die Datei `drill.glade` abgelegt werden.

Die notwendigen Module

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

Die graphische Schnittstelle mit LibGlade

Der Aufbau der graphischen Schnittstelle und die Einbindung der Bedienungsfunktionen mit LibGlade wird in analoger Weise wie im vorherigen Beispiel durchgeführt. Wir werden auf diesen Aspekt zurückkommen.

Im Python Programm definieren wir die globalen Variablen :

- `currentExercice`: Pointer auf das Widget, der die aktuelle Übung darstellt. Dieser wird in der rechten Seite des Anwendungsfensterns ***Drill*** abgelegt. Die Übungen sind ebenfalls von Glade aus geschrieben worden.
- `exerciceTree`: Pointer auf den Baum auf den linken Teil des Anwendungsfensters von ***Drill***.
- `label`: Pointer auf den label (`GtkLabel`). Dieser Label ist ein Notbehelf bei Nichtvorhandensein der Übung. Er wird auf der rechten Seite des Baumes gesetzt −− wo die Übungen hingehören −− und wir werden dort Namen der ausgewählten Übungen anzeigen.

Der Baum ist von LibGlade erzeugt worden, der Pointer ist mit folgendem Aufruf erhalten worden:

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

Wir benötigen ebenfalls den Pointer auf die horizontalen Fenster, genauer gesagt, den Pointer vom Container Widget (`GtkPaned`), dem horizontalen Fenster mit der Trennleiste. Die linke Hälfte stellt den Baum dar und die rechte Hälfte die Übungen, wo wir jetzt den label gesetzt haben:

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

Es empfiehlt sich, auch die **Bedienungsanleitung von GTK+ −−** über die Objekte `GtkLabel` und `GtkPaned` **−−** und die Quellen von Python `/usr/lib/python1.5/site-packages/gtk.py` zu studieren.

Das Widget `GtkTree`

Wir kommen jetzt zum wichtigsten Teil unseres Artikels, um einen Baum des Types `GtkTree` benutzen zu können.

Der Baum besteht aus aufeinanderfolgenden Aufrufen der Funktionen `addMathExercices()`, `addFrenchExercices()`, `addHistoryExercices()` et `addGeographyExercices()`. Diese Funktionen ähneln sich alle. Jede dieser Funktionen fügt eine neue Unterkategorie (Unterbaum) ebenso wie die Titel der Übung (die Items) hinzu:

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

Der Unterbaum

```
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
```

Um einen Unterbaum im existierenden Baum zu erzeugen, muss man zwei Dinge erzeugen: Einen Baum `GtkTree` und ein Element `GtkTreeItem`, das nach dem Unterbaum benannt ist. Anschließend ist das Element an der Wurzel des Baumes hinzugefügt worden **−−** unser Baum beinhaltet alle Kategorien **−−** dann setzen wir unseren Unterbaum auf das Element mit Hilfe der Methode `set_subtree()` auf. Schließlich ist der Event `select` an das Element anzuschließen. Damit wird die Funktion `selectSubtree()` aufgerufen sobald ein Kathegorie ausgewählte wird.

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)
```

Die Elemente tragen die gleichen Bezeichnungen wie die Namen der Übungen, hier im allgemeinen nur `Übung 1`, `Übung 2`, ... Bei jedem Element wird ein zusätzliches Attribut hinzugefügt, `id`. GTK+ bietet die

Möglichkeit jedes Objekt vom Typ `GtkObject` hinzuzufügen. Hierzu existieren zwei Methoden `set_data (key, value)` und `get_data (key)`, um das Attribut zu initialisieren und auf dessen Wert zurückzugreifen. Das Element wird anschließend zu seiner Kategorie hinzugefügt −− einem Unterbaum. Die Methode `show()` wird aufgerufen um das element anzuzeigen. Anschließend werden die Events `select` und `deselect` eingebunden. Das Event `deselect` wird eingesetzt, wenn das Element nicht mehr ausgewählt wird. Analog dazu wird die Methode `deselectTreeItem()` oder `selectTreeItem()` ausgeführt.

Die Bedienungsfunktionen

Wir haben drei Bedienungsfunktionen definiert `selectTreeItem()`, `deselectTreeItem()` und `selectSubtree()`. Diese Funktionen aktualisieren den Text des Labels −− in der rechten Zone −− mit dem Wert des Attributes `id`.

Schlusswort

Wir haben die Infrastruktur, in der wir die Übungen durchführen, aufgebaut −− genauso wie die neu entdeckten Widgets. Wir haben uns hauptsächlich mit dem Widget `GtkTree` beschäftigt und wie man Attribute zu den Widgets hinzufügen kann. Dieses Verfahren wird sehr oft benutzt, um in den Bedienungsfunktionen noch zusätzliche Informationen zu erhalten. Bis zum nächsten Artikel können Sie versuchen, das Spiel ***Couleur***, womit wir uns im Teil I beschäftigt haben, als eine Übung in ***Drill*** einzubauen.

Anhang: Der vollständige Sourcecode

```
#!/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 ()
```

2005−01−11, generated by lfparser_pdf version 2.51