Introduction à GTK en Python

E. Ramat et G. Quesnel

Université du Littoral - Côte d'Opale

13 septembre 2007

PyGTK qu'est ce ?

- *PyGTK* est une API de programmation d'interfaces graphiques.
- c'est un wrapper sur *GTK+* : the Gimp ToolKit ;
- disponible sur la plupart des plateformes : Unix/Linux, Windows et MacOS.
- cette bibliothèque se base sur les bibliothèques :
  - *gdk* (GIMP Drawing Kit) ou Win 32 ;
  - *glib* les fonctions de base (encapsulation de fonctions C : threads, chargement dynamique, ...), les outils de portabilité, ... ;
  - *pango* les fonctions de rendu du texte en mode graphique, prise en charge de l'internationalisation, ... ;
  - *ATK* interface d'accessibilité ;
- quelques exemples de projets GTK+ : *Inkspace*, *K-3D*, ...

Plan

Histoire

- *GTK+* est un toolkit développé pour Gimp 1.0 et repris par le projet GNU comme API de base pour le développement du projet Gnome.
- la philosophie de développement de *PyGTK* se base sur celle de *GTK+* : écrire le moins de code possible pour définir des interfaces graphiques simples d'utilisation.
- le nom des fonctions membres sont les mêmes que celle de *GTK+*.
- il existe un très grand nombre de « wrappers » pour *GTK+*. Les plus connus sont :
  - GTKmm, java-gnome, gtk2-perl, *PyGTK*
  - GtkAda, GTKKit

Les événements

- *GTK+* est un toolkit piloté par les événements ;
- votre programme “dort” tant qu'aucun événement ne se produit ;
- les événements ou `signal` dans les applications *PyGTK* se manipulent de deux manières :
  - par connexion d'un événement sur une méthode ;
  - par connexion d'un événement sur une fonction globale ;
- des paramètres peuvent être fournis à ces fonctions pour transmettre des information sur l'événement.

Les événements

```
handler_id = object.connect(name,fun,func_data)
```

Les événements

```
handler_id = object.connect(name,fun,func_data)
```

Commentaires

où `object` est l'instance d'une classe *PyGTK* qui a émit le `signal`, `name` le type de signal, `fun` une référence sur la fonction ou méthode à appeler et `func_data` les éventuels paramètres à passer à la fonction ou méthode.

Les événements

Quelques exemples de types d'événements :

- button_press_event : un bouton de la souris est enfoncé ;
- key_press_event : une touche du clavier est enfoncée ;
- scroll_event : la barre de défilement a été actionnée ;
- focus_in_event : le composant vient de récupérer le focus ;

La boucle événementielle

- un programme graphique utilise une boucle événementielle : tant que l'utilisateur ne fait pas d'action sur l'interface, celle-ci ne réagit pas.
- sur les systèmes X-Window, il existe une pile d'événements qui s'entassent et se dépilent au fur et à mesure des demandes de l'utilisateur ou du programme.
- les événements sont les mouvements et clic sur la souris, l'appui sur une touche, ldots

La boucle événementielle

- un programme graphique utilise une boucle événementielle : tant que l'utilisateur ne fait pas d'action sur l'interface, celle-ci ne réagit pas.
- sur les systèmes X-Window, il existe une pile d'événements qui s'entassent et se dépilent au fur et à mesure des demandes de l'utilisateur ou du programme.
- les événements sont les mouvements et clic sur la souris, l'appui sur une touche, ldots

La boucle événementielle

- un programme graphique utilise une boucle événementielle : tant que l'utilisateur ne fait pas d'action sur l'interface, celle-ci ne réagit pas.
- sur les systèmes X-Window, il existe une pile d'événements qui s'entassent et se dépilent au fur et à mesure des demandes de l'utilisateur ou du programme.
- les événements sont les mouvements et clic sur la souris, l'appui sur une touche, ldots

La boucle événementielle

- un programme graphique utilise une boucle événementielle : tant que l'utilisateur ne fait pas d'action sur l'interface, celle-ci ne réagit pas.
- sur les systèmes X-Window, il existe une pile d'événements qui s'entassent et se dépilent au fur et à mesure des demandes de l'utilisateur ou du programme.
- les événements sont les mouvements et clic sur la souris, l'appui sur une touche, ldots

Attention

Il ne faut surtout pas rester trop longtemps dans une réponse sinon les événements s'empileront dans la pile mais ne seront pas gérés au bon moment.

Le programme minimal

```
import pygtk
pygtk.require('2.0')
import gtk

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

  def main(self):
    gtk.main()

if __name__ == "__main__":
  application = Application()
  application.main()
```

Le programme minimal

```
import pygtk
pygtk.require('2.0')
import gtk

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

  def main(self):
    gtk.main()

if __name__ == "__main__":
  application = Application()
  application.main()
```

Attention

La fonction principale (*if __name__ = "__main__"*) créé une instance de la classe créatrice de l'application et lance la boucle

Le programme minimal

- le code précédent ne permet pas à l'application de se terminer correctement. Le bouton de fermeture de la fenêtre n'est pas actif ;
- il faut connecter l'événement de fermeture (delete_event) à une méthode de terminaison de la boucle événementielle ;

Le programme minimal

```
import pygtk
pygtk.require('2.0')
import gtk

class Application:
  def __init__(self):
    self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
    self.window.connect("delete_event", self.delete_event)
    self.window.show()

  def delete_event(self, widget, event, data=None):
    gtk.main_quit()
    return False

  def main(self):
    gtk.main()

if __name__ == "__main__":
  application = Application()
  application.main()
```

Plan

Les widgets

- un widget est un élément d'interface graphique, il peut désigner :
  - une fenêtre, une boîte de dialogue, . . . ;
  - un élément d'une fenêtre (liste déroulante, bouton, . . .) ;
-

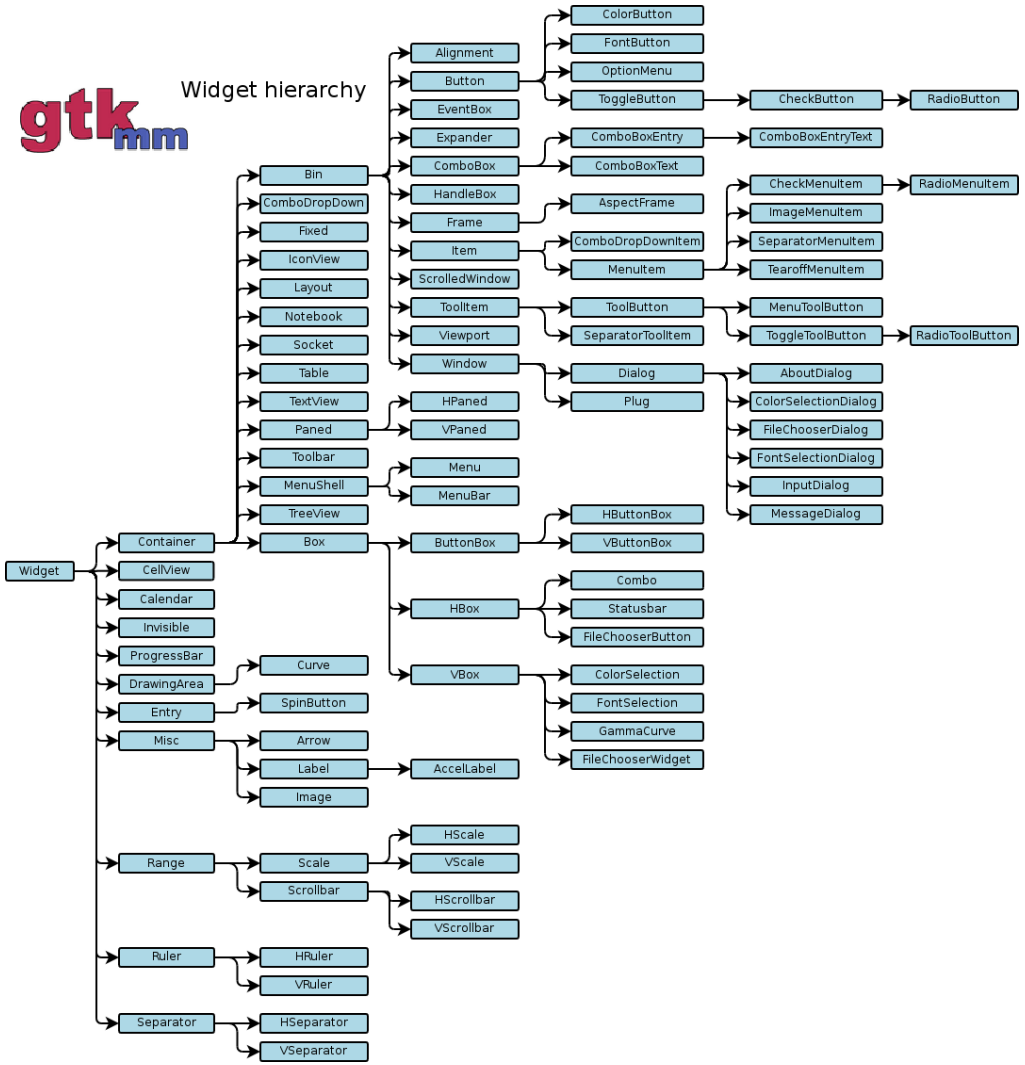

FIG.: Hiérarchie de classes GTK en C++

Fenêtre

La classe gtk.Window est le widget représentant une fenêtre dans l'environnement graphique. Cette classe possède les fonctions en relation avec le window manager, comme par exemple iconifier la fenêtre, l'enrouler, ...

Fenêtre

La classe gtk.Window est le widget représentant une fenêtre dans l'environnement graphique. Cette classe possède les fonctions en relation avec le window manager, comme par exemple iconifier la fenêtre, l'enrouler, ...

Attention

La fenêtre est un conteneur d'un seul widget !

Plan

Les conteneurs

- pour placer les widgets dans une fenêtre, il est nécessaire de les placer dans des conteneurs (Container) ;
- il existe deux types de conteneurs :
  - Conteneurs à un widget, héritent de gtk.Bin ;
  - Conteneurs multiples, héritent de gtk.Box ;
- puis d'organiser les widgets dans le conteneur : "pack" ;
- les conteneurs sont invisibles, ils permettent juste d'organiser le placement des widgets ;
- il existe deux types de widgets multiples : horizontaux (gtk.HBox)et verticaux (gtk.VBox) ;

Les conteneurs

- un conteneur peut contenir un autre conteneur.
- deux méthodes sont disponibles pour l'ajour d'un widget dans un conteneur :
  - pack_start : ajout à gauche dans un gtk.HBox ou en haut dans un gtk.VBox ;
  - pack_end : ajout à droite dans un gtk.HBox ou en bas dans un gtk.VBox.

Conteneurs : un exemple

```
import pygtk
pygtk.require('2.0')
import gtk

class Application:
  def __init__(self):
    self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
    self.window.connect("delete_event", self.delete_event)

    self.vbox = gtk.VBox()
    self.hbox = gtk.HBox()

    self.b1 = gtk.Button("b1")
    self.b2 = gtk.Button("b2")
    self.b3 = gtk.Button("b3")

    self.hbox.pack_start(b1, true, true, 0)
    self.hbox.pack_start(b2, false, false, 0)
    self.vbox.pack_start(b3, true, true, 0)
    self.vbox.pack_start(hbox, false, false, 0)
```

Conteneurs : un exemple

```
        self.window.add(self.vbox)
        self.window.show_all()

    def delete_event(self, widget, event, data=None):
        gtk.main_quit()
        return False

    def main(self):
        gtk.main()

if __name__ == "__main__":
    application = Application()
    application.main()
```

Conteneurs : un exemple

```
        self.window.add(self.vbox)
        self.window.show_all()

    def delete_event(self, widget, event, data=None):
        gtk.main_quit()
        return False

    def main(self):
        gtk.main()

if __name__ == "__main__":
    application = Application()
    application.main()
```

Information

À noter que plusieurs constructeurs de gtk.Button existent avec une icône, un texte, une image, /ldots

Conteneurs : un exemple

Les conteneurs

Deux types de conteneurs existent en *GTK+* suivant la capacité qu'il possède :

- Conteneurs à un widget, héritent de Gtk::Bin
  - Frame : un rectangle avec un label ;
  - Dialog : la boîte de dialogue minimale ;
  - Alignement : placement aligné sur une bordure ;
  - Window : la fenêtre ;
- Conteneurs multiple, héritent de Gtk::Container
  - VBox : une liste dynamiques d'espaces verticaux ;
  - HBox : une liste dynamiques d'espaces horizontaux ;
  - NoteBook : un widget à onglets ;
  - Table : un tableau (les widgets sont alignés sur une grille) ;

Information

Il existe bien sûr d'autres conteneurs.

Table

- le conteneur Table permet de définir une grille régulière afin de placer les widgets ;
- le constrcuteur de Table admet trois paramètres :
  - le nombre de lignes ;
  - le nombre de colonnes ;
  - le type de calcul de la taille des cases (homogène) :
    - True : la taille des cases est égale à la taille du plus grand widget ;
    - False : pour chaque colonne, la largeur des cases est égale à la largeur du widget le plus large ; idem pour les lignes ; la taille des cases n'est donc pas la même pour toutes les cases ;

Table

- l'ajout des widgets à la table est réalisée par la fonction attach(child, left_attach, right_attach, top_attach, bottom_attach, xoptions, yoptions, xpadding, ypadding) :
  - child : le widget à ajouter ;
  - left_attach, right_attach, top_attach et bottom_attach : le nombre de cases occupées en indiquant les index des cases (pour les 4 coins du widget) entre lesquels le widget sera placé (exemple : pour la case en haut à gauche et pour un widget occupant seulement une case, on donnera : 0,1,0,1);
  - xoptions et yoptions : ...
  - xpadding et ypadding : ...
- on peut spécifier l'espacement entre les lignes et les colonnes par les fonctions set_row_spacing(row,spacing) et set_col_spacing(col, spacing) ou set_row_spacings(spacing) et set_col_spacings(spacing) pour toutes les lignes et colonnes ;

Label

Le Label est un composant simple qui représente une chaîne de caractères dans une zone graphique.

```
if __name__ == "__main__":
    app = gtk.Window(gtk.WINDOW_TOPLEVEL)
    label = gtk.Label()
    label.set_markup("<i>hello</i> <u>world</u> <b>!</b>")
    app.add(label)
    app.show_all()
    gtk.main()
```

Label

Le Label est un composant simple qui représente une chaîne de caractères dans une zone graphique.

```
if __name__ == "__main__":
    app = gtk.Window(gtk.WINDOW_TOPLEVEL)
    label = gtk.Label()
    label.set_markup("<i>hello</i> <u>world</u> <b>!</b>")
    app.add(label)
    app.show_all()
    gtk.main()
```

Information

À noter que ce widget peut accepter une syntaxe simplifiée du langage HTML.

Zone de texte

Entry

Le widget Entry permet de saisir et d'afficher un texte dans une zone de texte à une ligne. La taille du texte peut être limitée.

```
if __name__ == "__main__":
    app = gtk.Window(gtk.WINDOW_TOPLEVEL)
    text = gtk.Entry()
    text.set_text("Hello␣World!")
    app.add(text)
    app.show_all()
    gtk.main()
```

Zone de texte

TextView

Le widget TextView fonctionne en modèle/vue/contrôleur où le modèle est un TextBuffer. On retrouve toutes les fonctions nécessaires comme la coloration syntaxique, la gestion des paragraphes et l'alignement du texte.

```
if __name__ == "__main__":
    app = gtk.Window(gtk.WINDOW_TOPLEVEL)
    text = gtk.TextView()
    text_buffer = text.get_buffer()
    text_buffer.insert(text_buffer.get_end_iter(),"Hello␣World!")
    app.add(text)
    app.show_all()
    gtk.main()
```

Zone de dessin

La classe DrawingArea est un widget sans spécificité. Il est laissé libre aux développeurs pour être hérité et amélioré. Les fonctions de dessins sont disponibles en récupérant une référence sur le Gdk::Window et le Gdk::GC.

```
class Application:
  def __init__(self):
    self.app = gtk.Window(gtk.WINDOW_TOPLEVEL)
    self.area = gtk.DrawingArea()
    self.area.set_size_request(100,100)
    self.area.connect("expose-event",self.area_expose)
    self.app.connect("destroy",lambda w: gtk.main_quit())
    self.app.add(self.area)
    self.area.show()
    self.app.show()
```

Zone de dessin

```
    def area_expose(self,area,event):
        width, height = self.area.window.get_size()
        self.gc = self.area.window.new_gc()
        self.area.window.draw_line(self.gc, 0, 0, width, height);
        self.area.window.draw_line(self.gc, 0, height, width, 0);

if __name__ == "__main__":
    app = Application()
    gtk.main()
```

Zone de dessin

```
    def area_expose(self,area,event):
      width, height = self.area.window.get_size()
      self.gc = self.area.window.new_gc()
      self.area.window.draw_line(self.gc, 0, 0, width, height);
      self.area.window.draw_line(self.gc, 0, height, width, 0);

if __name__ == "__main__":
    app = Application()
    gtk.main()
```

Information

L'événement *expose-event* se produit lors d'une demande d'affichage de la zone de dessin. L'attribut gc est un contexte graphique.

Les liens

Une petite liste de liens :

- *PyGTK* :
  `http://www.pygtk.org/docs/pygtk/index.html`

fin

...fin du cours...