Bevezetés a pyGTK és a Glade használatába

A pyGTK és a Glade bárki számára lehetővé teszi, hogy könnyen és gyorsan hozzon létre működő grafikus felületeket.

A pyGTK és a Glade szépsége abban rejlik, hogy megnyitották a profi minőségű, rendszerek közötti grafikus programok fejlesztésének útját azok számára, akik nem szeretnének túl elmélyülten foglalkozni a témával, mégis szükségük van egy grafikus felületre a programjukhoz. A pyGTK nemcsak a kezdők számára teszi lehetővé nagyszerű grafikus felületek írását, hanem a gyakorlott programozóknak is, hogy minden eddiginél gyorsabban hozzanak létre rugalmas, változtatható és hatékony felhasználói felületeket. Ha korábban vágytunk már arra, hogy szép megjelenésű felhasználói felületet hozzunk létre gyorsan és minél kisebb erőfeszítéssel, de nem vagyunk jártasak a grafikus felületek írásában, akkor érdemes tovább olvasnunk.

Ez a cikk annak a tanulási folyamatnak az eredménye, amelyen az Immunity CANVAS (➲ http://www.immunitysec.com/CANVAS) program megírása közben keresztülmentem. A grafikus felület fejlesztése közben megtanultak jelentős részét megtalálhatjuk a pyGTK GYK-ban a ➲ http://www.async.com.br/faq/pygtk/index.py?req=index címen. Egy másik leírás, amit minden bizonnyal sokat fogunk használni, ha elmerülünk a pyGTK rejtelmeibe, a ➲ http://www.gnome.org/~james/pygtk-docs címen található. Őszintén kijelenthetjük, hogy egy kisvállalat számára a pyGTK használata versenyképes előnyt jelenthet olyan más grafikus fejlesztőkörnyezetekkel szemben, mint amilyen például a C. Ennek a cikknek az elolvasása után remélhetőleg mindenki képes lesz arra, hogy az összes nyelv közül a legkönnyebben elsajátítható Python segítségével összeüssön egy GUI-t.

Összehasonlításként: a CANVAS grafikus felülete az alapoktól kiindulva körülbelül két hét alatt készült el, anélkül, hogy a pyGTK-ról előzetes ismeretekkel rendelkeztem volna. Ezt követően a GTK v1-ről GTK v2-re egy nap alatt ültettem át (erről később még lesz szó), és most már mind a Microsoft Windows-, mind a Linux-rendszereken működőképes.

A pyGTK rendszereket áthidaló jellege

Egy tökéletes világban soha nem lenne szükségünk arra, hogy a Linuxon kívül bármilyen más rendszerre programot kelljen fejlesztenünk, s ezt kedvenc rendszercsomagunkon tehetnénk meg. A valós világban támogatnunk kell a Linux különböző változatait, a Windowst, a Unixot vagy bármilyen más operációs rendszert, ha az ügyfél úgy kívánja. A grafikus eszközkészlet kiválasztása nem kis mértékben azon múlik, hogy a megbízónk által használt felületen melyik élvez nagyobb támogatást. Napjainkban a Python fejlesztőeszközként történő választása bármilyen új próbálkozásnál már természetessé kezd válni, ha a fejlesztés gyorsasága nagyobb jelentőséggel bír, mint a futtatási sebesség. Ezzel el is érkezünk a Python GUI fejlesztőeszközének választási lehetőségeihez, amit a wxPython, a Tkinter, a pyGTK és a Python/Qt képez.

Szem előtt tartva a tényt, hogy nem vagyok a grafikus felületek fejlesztésének szakértője, érzéseim alapján a következő érvek szólnak a pyGTK választása mellett: a wxPython már hosszú ideje elérhető és vonzó felületek létrehozásának lehetőségét kínálja, de nehéz munkára fogni és használni, különösen egy kezdőnek. Nem beszélve arról, hogy mind a Linux, mind a Windows felhasználói nagyméretű bináris állományt kénytelenek letölteni és telepíteni. A Qt-nek – bár Linux alá ingyenes – Windows alatti használata már jogdíjhoz kötött, s ez a tulajdonsága számos olyan kisvállalkozásnál kizáró tényező lehet, amelyek programjukat mindkét operációs rendszer alatt terjeszteni kívánják.

A Tkinter az első Python GUI-fejlesztőkészlet, szinte minden Linux-terjesztésnek része. Emellett csúnya, és a Tk-nak a Python-programokba történő beágyazását igényli, ami azt az érzést kelti, hogy visszafelé haladunk. Egy kezdő számára nagyon fontos dolog, hogy amennyire csak lehetséges, a grafikus felületet elválassza magától a programtól. Ez lehetővé teszi annak elkerülését, hogy a GUI szerkesztésekor a programon is egy sereg változtatást végre kelljen hajtani, vagy valamilyen módosítást bele kelljen építeni.

A fenti okokból kifolyólag a leginkább a pyGTK az, amire szükségünk van. A pyGTK gondosan különválasztja a programot és a grafikus felületet. A ***libglade*** használatával maga a GUI XML-fájlként tárolódik, amit tovább szerkeszthetünk, többféle változatot készíthetünk belőle, vagy bármit megtehetünk vele, mivel nincs összeépítve a programkóddal. A Glade GUI-készítő programként való használatával gyorsan szerkeszthetünk programfelületeket. A fejlesztési idő olyan rövid, hogyha a különböző felhasználók eltérő felületeket szeretnének maguknak, mindnyájuk igényeit könnyen kielégíthetjük.

A GTK és a pyGTK változatai

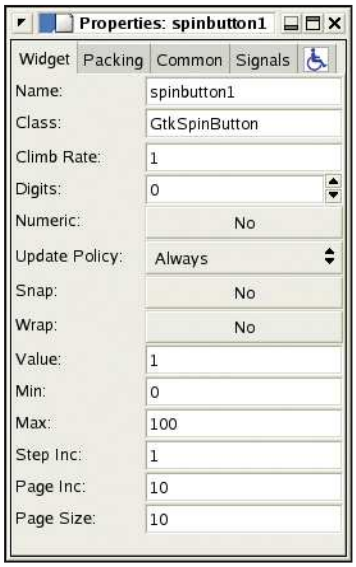
A GTK-nak két változata érhető el, az 1-es és a 2-es, ezért mielőtt GUI-fejlesztő projektünkbe belevágnánk, el kell döntenünk néhány dolgot a fejlesztéssel és karbantartással kapcsolatban. Előfordulhat, hogy le kell töltenünk a Gladev2-változatát, vagy telepítenünk kell a GTK fejlesztői csomagjait ahhoz, hogy a GTK v2 ***libglade*** programkönyvtárat lefordíthassuk. Biztosíthatom olvasóinkat, ez megéri a fáradságot. A GTK v2 számos előnyt kínál: szebb kinézetet, telepítőt a Windowshoz a Python 2.2-es változatával, a hozzáférhetőséggel kapcsolatos bővítményeket, amely vak felhasználók számára teszi lehetővé az alkalmazások elérését. Ráadásul a 2-es változat a frissebb rendszercsomagok közül sokban előre telepítve érkezik, bár még mindig lehetséges, hogy a fejlesztői RPM-csomagokat és a legfrissebb pyGTK-t magunknak kell telepítenünk.

A GTK v2 és így a pyGTK v2 is nyújt néhány, egy kicsit összetettebb elemkészletet. Egy nagy tudású GUI-mester kezében ezek az eszközök csodálatos dolgokra képesek, de a kezdőt csak összezavarják. Igaz ugyan, hogy néhány kódrecept úgy állítja be őket, mintha a GTK v1-es eszközeinek másolataiként – ha egyszer megtanultuk a használatukat – kezelhetőek lennének. Például amikor a GTK v1-es változatával a CANVAS számára a teljes grafikus felületet kifejlesztettem, GTK v2-esben át kellett terveznem (ez pontosan egy napot vett igénybe). Ügyfeleim

linuxos gépeiről hiányzott ugyan a GTK v1 támogatása, de a GTK v2 telepítése egészen egyszerű volt. Az egyetlen kivétel a Ximian Desktop, ami határozottan megkönnyíti a pyGTK és a GTK v1 telepítését. Vagyis ha a teljes ügyfélkörünk ezt használja, esetleg kitarthatunk a GTK v1-es mellett. Egy dolgot azonban jól meg kell jegyeznünk: létezik Python-parancsfájl arra a célra, hogy a Glade v1-esben írt projektünket Glade v2-es változatra alakítsuk át, de fordítva már nem megy. Így ha mindkettőre szükségünk van, először a Glade v1-esben fejlessszünk, ezután alakítsuk át azt és hangoljuk össze a különbségeket.

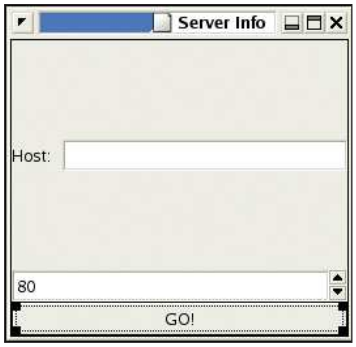
Bevezetés a Glade v2-esbe

A Glade és a ***libglade*** használata mögött az az elgondolás húzódik meg, hogy grafikus felületünket időpocsékolás kódolással létrehoznunk. Csak ülni és mást se csinálni, mint megadni az egyes eszközök helyét, színét és alapértelmezett beállításait a Python-fordítónak – ez bizony igen időrabló tevékenység. Akinek volt már dolga Tcl/Tk-programozással, biztos, hogy napokat töltött el ilyesmivel. És nem csupán ez, de egy kóddal megírt GUI módosítása is tekintélyes vállalkozás lehet a rá fordított időmennyiség tekintetében. A Glade és ***libglade*** esetében a kód írása helyett XML-fájlokat hozunk létre, valamint olyan kódhivatkozásokat, amelyek egy nyomógombot, beviteli mezőt, vagy kimeneti szövegtárolót tartalmazó fájlra mutatnak.
A munka elkezdéséhez – ha még nem rendelkeznénk vele – szükségünk van a Glade v2-esre. Amennyiben már megvan, akkor is jól jöhet a legfrissebb változata. Ha korábban már telepítettük a GTK v2 fejlesztői csomagokat (a `-devel` RPM-csomagokat), a Glade v2 letöltése és telepítése sem okozhat gondot. Mindenesetre a legtöbb, GUI-fejlesztésben még kevéssé jártas felhasználó számára félelmetesen üres a kezdőablak. Programunk fejlesztésének megkezdéséhez kattintsunk a ***Window*** ikonra. Most egy nagy üres ablakot kell látnunk a képernyőn.
Az egyik fontos dolog a grafikus felületek fejlesztésével kapcsolatban az, hogy alapvetően kétféle objektummal dolgozunk: eszközökkel, ilyenek a címkék, a beviteli szövegdobozok és az egyéb látható dolgok, és ezeknek az eszközöknek a tárolóival. A legvalószínűbb, hogy a tárolók három fajtája (a vízszintes doboz, a függőleges doboz és a tábla) közül egyet fogunk használni. Az összetett külső előállításának a legkönnyebb módja az, ha a kívánt sorrendben egymásba ágyazzuk őket. Kattintsunk például a vízszintes doboz ikonjára. A ***window1*** vonalkázott területére kattintva három újabb területet szúrunk be, amelyekre eszközöket helyezhetünk.
Most ennek a három területnek bármelyikét kiválaszthatjuk, és tovább oszthatjuk egy vízszintes dobozzal. Ha nem vagyunk elégedettek az eredménnyel, visszatérhetünk és törölhetjük, kivághatjuk és beilleszthetjük, vagy megváltoztathatjuk a dobozok számát a ***Properties*** (tulajdonságok) menüből (erről még később lesz szó).
Ezekből az elemi alkotórészekből szinte bármilyen elrendezés előállítható. Most, hogy már rendelkezünk egy kiinduló beosztással, olyan eszközökkel tölthetjük fel, amelyeknek már van valamilyen rendeltetésük is. Én most egy címkével, egy szövegbeviteli dobozzal, egy léptetőgombbal (spinbutton) és egy közönséges gombbal töltöm fel. Elsőre nem valami szép látvány. Ne feledjük, hogy a GTK a kész ablakot a megjelenítéskor önműködően méretezi, ezért mindent annyira szorosan helyeztem el, amennyire csak lehetett. Amikor a felhasználó elhúzza az ablak sarkát, az is önműködően változtatja a méretét. Ezeket a beállításokat a ***Properties*** (tulajdonságok) ablakban tehetjük meg (váltsunk a Glade főablakára és kattintsunk a ***View®Show Properties*** menüpontra). Az ablakban a különböző eszközök más-más tulajdonságait változtathatjuk meg. Ha például a léptetőgomb van a fókuszban, akkor az *1. képen* látható lehetőségek állnak rendelkezésünkre.



1. kép
A Glade-ben az eszközök jellemzőinek megváltoztatására szolgáló felület az eszközök függvényében változik

A ***Value*** (érték) tulajdonsággal a léptetőgomb megjelenéskori alapértelmezett értékét módosíthatjuk. Fontos a ***Max*** érték helyes beállítása is. Gyakori hiba a ***Value*** tulajdonság valamilyen magas értékre történő beállítása anélkül, hogy a ***Max*** értéket is megfelelően hozzáigazítanánk. Ennek az az eredménye, hogy a léptetőgomb először a beállított ***Value*** értéket mutatja, de amikor a felhasználó állítani próbálja, visszaáll a ***Max***-értékre – ami meglehetősen zavaró jelenség. Esetünkben ezt az eszközt a TCP-kapu beállítására fogom használni, így 65535-re állítom be, a legkisebb értéket pedig 1-re és az alapértelmezést 80-ra.
Ezután a ***label1*** címkére fókuszálok, és a ***Host:*** feliratot adom neki. A ***window1*** feliratra kattintva a Glade főablakában az egész ablakunk a fókuszba kerül, amelynek így szintén beállíthatjuk a jellemzőit. Ezt úgy is megtehetjük, hogy láthatóvá tesszük az eszközfa (widget tree) ablakát, és ebben kattintunk a ***window1***-re. A nevének ***serverinfo***-ra való változtatásával és a címnek (title) ***Server Info*** értéket adva a programunknak megfelelően állíthatjuk be az ablak fejlécét és az eszköz Glade-beli magas szintű nevét.



2. kép
A Glade-ben a grafikus felület nem teljesen olyan, mint a lefordított, úgyhogy ne aggódjunk a Host: területe miatt

Ha visszatérünk az eszközfanézetre és a ***hbox1***-re kattintunk, növelhetjük a térközt a ***Host:*** felirat és a szövegbeviteli doboz között. Ezzel egy kicsit tovább csinosíthatjuk az elrendezést. Kész grafikus felületünk a *2. képen* látható.
Rendes körülmények közt mindez csak néhány percig tart. Egy kis gyakorlás után látni fogjuk, hogy a Glade használatával még a legösszetettebb GUI létrehozása is csak perceket vesz igénybe. Hasonlítsuk ezt össze azzal az idővel, amit a Tk-parancsok begépelésére fordítanánk, ha ugyanezt akarnánk vele létrehozni.
Ez a felület természetesen még nem csinál semmit. Ahhoz még meg kell írnunk azt a Python-kódot, ami a ***.glade*** fájlt betölti és elvégzi a tényleges műveleteket. Most már valójában két Python-fájlt írok minden Glade által vezérelt projekthez:

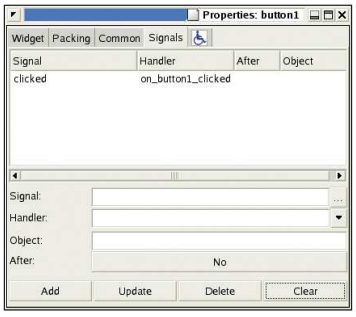
az egyik a GUI-t kezeli, a másik pedig teljesen független ettől a grafikus felülettől. Ezzel a módszerrel egyszerűvé válik a GTK v1-esről a GTK v2-esre vagy akár egy másik GUI-eszközre történő átültetés.

A Python-program létrehozása

Először is a lehetséges változatkülönbségekből adódó gondokat kell elhárítanunk. Én az alábbi kódot használtam, bár a GYK-ban említett bejegyzések némelyike hasonlóan működik:

```
#!/usr/bin/env python

import sys

try:
 import pygtk
  #közöljük a  pyGTK-val, hogy ha lehetséges,
  #a GTKv2-t használnánk
  pygtk.require("2.0")
except:
  #Néhány rendszercsomagban van GTK2,
  #de pyGTK nincs
  pass

try:
  import gtk
  import gtk.glade
except:
  print "You need to install pyGTK or GTKv2 ",
  print "or set your PYTHONPATH correctly."
  print "try: export PYTHONPATH=",
  print "/usr/local/lib/python2.2/site-
        ➥packages/"
  sys.exit(1)

#Most már rendelkezünk beolvasott gtk-val
#és gtk.glade-del,
#Øs az is biztos, hogy a GTK v2 fut
```

Most `appGUI` néven létrehozunk egy GUI-osztályt. Mielőtt ezt megtennénk, meg kell nyitnunk a ***button1*** tulajdonságait, és létre kell hoznunk egy eseményt. Ehhez kattintsunk a három pontra, görgessünk a ***clicked*** feliratra, válasszuk ki, majd kattintsunk az ***Add***-ra.



3. kép
Az eseménykezelő létrehozása után

A *3. képen* látható az eredmény, amit kapnunk kell. Ezzel a gombra való kattintás egy `signal_autoconnect` eseményt hoz létre, ami az egyik függvényünk (`button1_clicked`) hívását eredményezi. A listában látható a többi, kezelésre váró lehetséges esemény is. Minden eszköz különböző várható jelzésekkel rendelkezhet. Például egy szövegváltozási jelzés hasznos lehet egy szövegbeviteli dobozon, de egy gombon értelmetlen, mert annak a felirata nem szerkeszthető.
A program elindításával és a `gtk.mainloop()` futtatásával működésbe lép a szerkezet. A különböző eseménykezelőknek eltérő számú értékekkel kell rendelkezniük. A kattintás eseménykezelője csak egyetlen értékkel bír: annak az eszköznek a nevével, amin a kattintás történt. Amíg a főablakon vagyunk, adjuk hozzá a ***destroy*** (megsemmisít) eseményt, így ha ezt az ablakot bezárjuk, a program futása véget ér. Ne feledkezzünk meg Glade projektünk mentéséről sem!

```
Glade project.
class appgui:
  def __init__(self):
    """
    Ebben az initben a Server Info
    főablakát jelenítjük meg
    """
    gladefile="project1.glade"
    windowname="serverinfo"
    self.wTree=gtk.glade.XML
(gladefile,windowname)
    # csak két regisztrálandó visszahívásunk
    # van, de akárhányat regisztrálhatnánk,
    # vagy használhatunk egy különleges
    # osztályt a visszahívások önműködő
    # regisztrálására, ha egy értéket
    # szeretnénk átadni, az alábbi tuple-hoz
    # hasonlót használnánk (??):
    # dic = { "on button1_clicked" :
    # (self.button1_clicked, arg1,arg2) , ...


dic = { "on_button1_clicked" :
             self.button1_clicked,
             "on_serverinfo_destroy" :
             (gtk.mainquit) }
    self.wTree.signal_autoconnect (dic)
    return

#####visszahívások
  def button1_clicked(self,widget):
    print "button clicked"

# Így indítjuk el a programot...
app=appgui()
gtk.mainloop()
```

Ha a pyGTK telepítését forráskódból végeztük, nagyon fontos, hogy a `PYTHONPATH` környezeti változó a ***/usr/local/lib/python2.2/site-packages/*** helyre mutasson, hogy a rendszer megtalálja a pyGTK-t. Ne mulasszuk el a ***project1.glade*** fájlnak a pillanatnyi könyvtárunkba való másolását sem. A ***GO!*** feliratra kattintáskor egy csinos üzenetnek kell megjelennie a terminálablakban.
Ahhoz, hogy a program valami érdemlegeset is csináljon, valahogy meg kell adnunk, hogy melyik gépet és kaput használjuk. A következő kódrészlet a `button1_clicked()` függvénybe másolva elvégzi ezt a feladatot:

```
host=self.wTree.get_widget("entry1").get_text()
port=int(self.wTree.get_widget(
  "spinbutton1").get_value())
if host=="":
  return
import urllib
page=urllib.urlopen(
```

```
  "http://"+host+":"+str(port)+"/")
data=page.read()
print data
```

Ha most a *GO!*-ra kattintunk, programunknak el kell látogatnia egy távoli honlapra, be kell olvasnia az oldal tartalmát, és ki kell azt listáznia a terminálablakra. Tovább finomíthatjuk az eredményt azáltal, hogy még több sort hozunk létre a `hbox`-ban, és egyéb eszközöket – például menüsort – adunk a programhoz. Próbálkozhatunk azzal is, hogy `table` elemet használunk az egymásba ágyazott `hbox`-ok és `vbox`-ok helyett, ami az elemek egymáshoz való igazításával gyakran szebb elrendezéshez vezet.

A TextView eszköz

Talán mégsem arra vágyunk, hogy az összes szöveg a terminálablakban jelenjen meg, nem igaz? Valószínűbb, hogy egy másik eszközön vagy ablakon szeretnénk látni. A GTK v2 erre a célra a `TextView` és `TextBuffer` eszközöket használja. A GTK v1 egy könnyen érthető eszközt kínál, amit egyszerűen GTKText-nek neveznek. Adjunk egy ***TextView***-t a projektünkhöz, és az eredményt írassuk ki ebbe az ablakba. Látni fogjuk, hogy egy ***scrolledwindow*** elem jön létre, és arra vár, hogy beépítsük. A ***TextBuffer*** létrehozásához a lenti sorokat adjuk hozzá `init()` függvényünkhöz és fűzzük hozzá ***TextView*** elemünkhöz. A GTK v2 használatának egyik nyilvánvaló előnye, hogy ugyanazt az átmenetitár-tartalmat két különböző módon is megnézhetjük. Ha akarjuk, a ***scrolledwindow1 Properties*** ablakában módosíthatjuk az ablak tulajdonságait, nagyíthatjuk például a méretét, hogy megfelelő hely álljon rendelkezésre a tartalom megjelenítéséhez:

4. kép
A GO!-ra kattintva betöltődik a weboldal és megjelenik a TextView-ban

```
self.logwindowview=self.wTree.get_widget
➥("textview1")
self.logwindow=gtk.TextBuffer(None)
self.logwindowview.set_buffer(self.logwindow)
```

A `button1_clicked()` függvényünkben a `print` utasítást cseréljük ki az alábbi sorokra:

```
self.logwindow.insert_at_cursor(data,len(data))
```

Most már akárhányszor a *GO!*-ra kattintunk, az eredmény az ablakunkban jelenik meg. A főablakunkat vízszintes panelekkel felosztva, ha kedvünk tartja, ezt az ablakot átméretezhetjük (*4. kép*).

A TreeView és a List eszközök

A GTK v1-es változattal ellentétben a GTK v2-esben a fa (tree) és a lista (list) alapvetően ugyanazt jelenti, a különbség a használt tárolási módban van. Egy másik fontos fogalom a `TreeIter`, ami egy lista vagy fa bizonyos sorára mutató pointer tárolására használt adattípus. Önmagában semmilyen hasznos eljárást nem kínál, azaz a `++` műveletet nem használhatjuk arra, hogy a fa vagy lista elemein lépegessünk. Ellenben ha bármikor egy fa bizonyos elemére akarunk hivatkozni, a `TreeView` eljárásai ezt az értéket kapják meg. Erre látunk példát a következő kódrészletben:

```
import gobject
self.treeview=[2]self.wTree.get_widget
➥("treeview1")
self.treemodel=gtk.TreeStore
➥(gobject.TYPE_STRING, gobject.TYPE_STRING)
self.treeview.set_model(self.treemodel)
```

Ezzel egy kétoszlopos famodellt határoztunk meg, amelynek az oszlopai egy-egy karakterláncot tartalmaznak. A következő kód címet ad az oszlopoknak:

```
self.treeview.set_headers_visible(gtk.TRUE)
renderer=gtk.CellRendererText()
column=gtk.TreeViewColumn
➥("Name",renderer, text=0)
column.set_resizable(gtk.TRUE)
self.treeview.append_column(column)
renderer=gtk.CellRendererText()

column=gtk.TreeViewColumn
➥("Description",renderer,text=1)
column.set_resizable(gtk.TRUE)
self.treeview.append_column(column)
self.treeview.show()
```

A következő függvényt arra használhatjuk, hogy kézzel vihessünk adatokat a faszerkezetbe:

```
def insert_row(model,parent,
➥firstcolumn,secondcolumn):
    myiter=model.insert_after(parent,None)
    model.set_value(myiter,0,firstcolumn)
    model.set_value(myiter,1,secondcolumn)
    return myiter
```

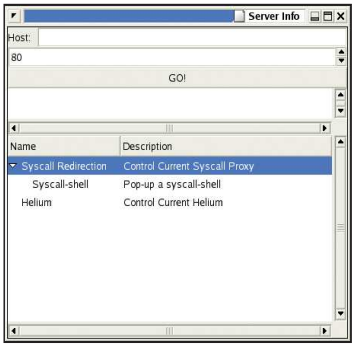
Íme egy példa a függvény használatára. Ne felejtsük el a ***treeview1***-et hozzáadni a glade-fájlunkhoz, majd menteni és átmásolni a helyi könyvtárunkba.

```
model=self.treemodel
insert_row(model,None,´Helium´,´Control
➥Current Helium´)
syscallIter=insert_row(model,None,´Syscall
➥Redirection´,´Control Current
➥Syscall Proxy´)
insert_row(model,syscallIter,
➥´Syscall-shell´,´Pop-up a syscall-shell´)
```

A *5. képen* látható pillanatkép mutatja az eredményt. Mint látható, a `TextView`-t egy `TreeView`-val helyettesítettem. A lista hasonló módon kezelhető, azzal a különbséggel, hogy `TreeStore` helyett `ListStore`-t használunk, és az `insert_after()` függvény helyett nagy valószínűséggel a `ListStore.append()` függvényt.

Párbeszédablakok használata

A párbeszédablak egy egyszerű ablaktól a leginkább abban különbözik, hogy valamilyen értéket kell visszaadnia. Egy párbeszédablak létrehozásához kattintsunk a ***dialog box*** gombra, majd adjunk nevet az alkotóelemnek. Ezután a kódunkban a `[3]gtk.glade.XML(glade-file,`*`párbeszédablak_neve`*`)` függvénnyel adjuk át. Ezt követően hívjuk meg a `get_widget(`*`párbeszédablak_neve`*`)` függvényt, hogy azonosítót rendeljünk ehhez az eszközhöz, majd hívjuk a `run()` eljárását. Ha az eredmény `gtk.RESPONSE_OK`, akkor a felhasználó az ***OK***-ra kattintott.
Ha nem, akkor vagy a ***Cancel***-t választotta, vagy bezárta az ablakot. Ebben az esetben a `destroy()` függvénnyel az ablakot megsemmisítjük és eltüntetjük.
A párbeszédablakok használatának egyik hátulütője, hogyha kivétel történik az elem `destroy()` eljárásának meghívása előtt, akkor egy semmire nem válaszoló párbeszédablak maradhat a képernyőn, ami igen zavaró lehet a felhasználó számára. Figyeljünk arra, hogy közvetlenül az adott válasz és az összes, a párbeszédablaktól várt adat beérkezése után hívjuk meg az eszköz `destroy()` eljárását.



5. kép
Példa a TreeView-ra két oszloppal

Az input_add() és a gtk.mainiteration() használata a foglalatok kezelésére

Egy nap talán éppen olyan pyGTK-programot írunk, ami foglalatokat (sockets) használ. Ha így lenne, vigyázzunk arra, hogy amíg az események kezelése folyamatban van, a programunk semmi mással nem foglalkozik. Amikor például egy `socket.accept()` eljárás végrehajtására várunk, zavaró lehet egy semmire nem válaszoló programot nézni. Ehelyett használjuk inkább a `gtk.input_add()` függvényt a szükséges számú foglalalt létrehozására, amelyek az eseményeket beolvassák a GTK belső listájára. Ez lehetővé teszi, hogy visszahívásokat adjunk meg, amelyek a foglalatokon keresztül érkező bármilyen adatot kezelni képesek.
Az ebben a módszerben rejlő egyik csapda az, hogy az esemény folyamán gyakran frissíteni akarjuk az ablakainkat, és ez szükségessé teszi a `gtk.mainiteration()` meghívását. Viszont ha akkor hívjuk meg a `gtk.mainiteration()` függvényt, amikor az éppen végrehajtás alatt áll, a program lefagy. A CANVAS esetén ezt én úgy oldottam meg, hogy a `gtk.mainiteration()` minden hívását egy olyan vizsgálattal vettem körül, ami biztosította, hogy nem történik rekurzív függvényhívás. Ellenőrzöm a folyamatban lévő eseményeket – ilyen a `socket.accept()` – minden olyan esetben, amikor naplóbejegyzést készítek. Naplózó függvényem a következőképpen fejeződik be:

```
def log(self,message,color):
"""
a naplózóablakra naplózza az eseményt most
éppen nem veszi figyelembe a szín értékét
"""
        message=message+"\n"
        self.logwindow.insert_at_cursor
        ➥(message,len(message))
        self.handlerdepth+=1
        if self.handlerdepth==1 and
        ➥gtk.events_pending():
            gtk.mainiteration()
        self.handlerdepth-=1
        return
```

Egy GUI-nak GTK v1-esből GTK v2-esbe történő átírása

A pyGTK GYK-nak az a része, ami a programok GTK v1-esből GTK v2-esbe való átültetéséről szól, egyre jobban közelít a befejezéshez. Ennek ellenére óvatosaknak kell lennünk azok miatt a nehézségek miatt, amikkel esetleg szembetalálkozhatunk. Az kézenfekvő, hogy minden `GtkText` elemet `Gtk.TextView` elemmel kell helyettesítenünk. Az ehhez kapcsolódó kódot a GUI-ban szintén meg kell változtatnunk, hogy hozzáigazítsuk a végrehajtott cseréhez. Ehhez hasonlóan az összes listát vagy fát, amit a GTK v1-esben létrehoztunk, újra létre kell hoznunk. Meglepetésként érhet minket, hogy a párbeszédablakokkal kapcsolatos teendőket is újra el kell végeznünk, hogy a GTK v2-es sokkal szebben mutató formátumára alakítsuk át őket.
Néhány olyan szövegbeli változtatást is el kell végeznünk, mint a GDK ***gtk.gdk***-ra, vagy a ***libglade gtk.glade***-re történő cseréje. A legtöbb esetben ez egy egyszerű keresés–csere művelettel elvégezhető. Továbbá a `GtkText.insert_defaults` használatos a `GtkTextBuffer.insert_at_cursor()` helyett és a `radiobutton.get_active()` a `radiobutton.active` helyett. A Glade v1-es fájlunkat Glade v2-essé alakíthatjuk a ***libglade*** csomagjának Python-parancsfájljával. Ez a grafikus felület tekintetében megteszi az első lépéseket, de előfordulhat, hogy be kell töltenünk a Glade v2-est, és újra elvégeznünk még néhány beállítást, még mielőtt a kódot átültetnénk.

Befejező megjegyzések

- Ne felejtsük el, hogy a kivágás–beillesztés műveleteket használhatjuk a Glade eszközfáján. Ez felgyorsítja és megkönnyíti az áttervezést.
- Oldjuk fel (unset) a lehetséges elhelyezkedéseket a ***Properties*** ablakban, hogy az indulókép ne nézzen ki olyan furcsán.
- Ha olyan kérdésünk van, amiről azt gondoljuk, hogy más is szembekerülhet vele, vegyük fel a pyGTK GYK-ba. A Gnome IRC-kiszolgálón működik egy hasznos ***#pygtk*** csatorna. Nem lettem volna képes a CANVAS megírására a csatornán keresztül kapott segítség nélkül, amelyek közül különösen ***John Henstridge***-é volt értékes számomra. Nagy elismerése a nyílt forrás közösségének, hogy a kezdők gondjainak megoldása érdekében a legfontosabb fejlesztők is gyakran elérhetők. A kidolgozott bemutató kód megtalálható az 52. CD Magazin/pyGTK könyvtárában, illetve letölthető a ➲ ftp.ssc.com/pub/lj/listings/issue113/6586.tgz címről.

Linux Journal 2003. augusztus, 113. szám

Dave Aitel
A New Yorkban működő Immunity, Inc biztonsági tanácsadó társaság alapítója. A CANVAS az Immunity betöréstesztelő és -felderítő fejlesztői keretprogramja, ami teljes egészében Pythonban, a pyGTK használatával készült.