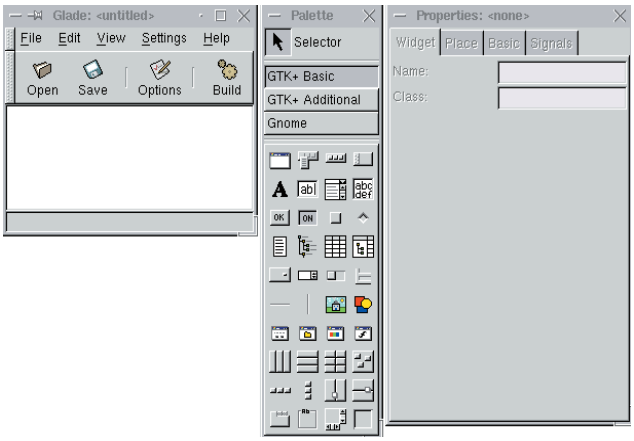
Felhasználói felületek létrehozása Glade-del

Hogyan használjuk a gnome-python libglade-jét Python-alapú grafikus alkalmazások fejlesztéséhez?

A Glade a Gtk+ eszközkészlet GUI-készítője (GUI = Graphical User Interface – grafikus felhasználói felület). A Glade segítségével kényelmesen gyárthatunk felhasználói felületeket és hozzá forráskódot, valamint ezekhez a felületekhez a visszahívó (callback) függvények vázát is létre tudjuk hozni.
A *libglade* programkönyvtár lehetővé teszi a programok számára, hogy a Glade-projektfájlokban megadott elemszerkezetet könnyen megjelenítsék. Ennek része a projektfájlban megnevezett visszahívók csatolása a program által nyújtott visszahívó programrészekhez.
James Henstridge a fenntartója a libglade és a gnome-python csomagnak, mely a Pythont a Gtk+ eszközkészlettel és a Gnome felhasználói felülettel köti össze, valamint magával a libglade-del. Ha Python-alapú grafikus alkalmazásainkban libglade-et használunk, a fejlesztési és a karbantartási idő jelentősen rövidül.
A cikkben szereplő összes példaprogram Glade 0.5.11, gnome-python 1.0.53 és Python 2.1b1 segítségével készült, Mandrake Linux 7.2 rendszeren.



1. kép A Glade indítása

A Glade futtatása

Indítás után a Glade három ablakot jelenít meg (lásd *1. kép*). Az alkalmazás főablaka mutatja a pillanatnyi Glade-projektfájl tartalmát, azaz a projektfájlban meghatározott ablakok és párbeszédablakok listáját.
A palettaablak mutatja a Glade által támogatott Gtk+ és Gnome-elemeket. Amikor egy elemet szerkesztésre kijelölünk, a tulajdonságablak jeleníti meg az elem tulajdonságainak pillanatnyi értékeit.
A palettaablak három csoportra osztja a Glade által támogatott elemeket: a „GTK+ Basic” (Alap) a leggyakrabban használt Gtk+ elemeket, a „GTK+ Additional” (Kiegészítő) a ritkábban alkalmazott elemeket tartalmazza, például a vonalzót és a naptárat. A Gnome-elemek a GNOME UI programkönyvtárból származnak.
A tulajdonságablak négylapos füzetben ábrázolja az elem tulajdonságait. A *Widget* lap az elem nevét és az elemosztályra jellemző tulajdonságokat jeleníti meg. Amikor az elem egy kényszerfeltétel-alapú tároló (például GTKTable vagy GTKVBox) belsejében helyezkedik el, a *Place* lap azokat a tulajdonságokat mutatja meg, amelyek az elem elhelyezkedését befolyásolják a tárolóban – máskülönben ez a lap üres. A *Basic* oldal olyan alapvető tulajdonságokat tartalmaz, mint például a szélesség és a magasság, ezekkel minden elem rendelkezik. Végül a *Signals* oldal segítségével adható meg, hogy az elem milyen jelzéseket adjon ki, és ezekhez a jelzésekhez megadhatjuk a kezelő függvényeket.

Elemszerkezetek létrehozása

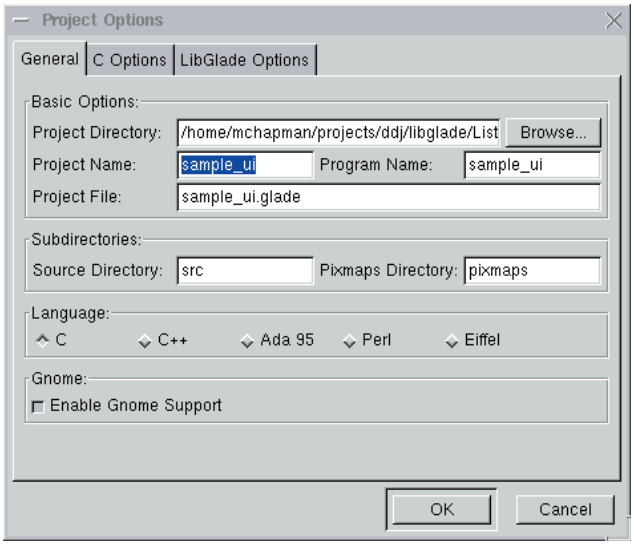
Az elemszerkezetek létrehozásának folyamata a Glade-ben hasonlóképpen zajlik, mint a Visual Basicben. A szerkezet kiindulópontja a felső szintű ablak vagy párbeszédablak. Az elemek a felső szintű ablakban helyezhetőek el úgy, hogy előbb a Glade palettaablakából kiválasztjuk őket, azután a tároló olyan részére kattintunk, ahol a célkereszt megjelenik.

Jelzéskezelők létrehozása

A Signals lap a Glade tulajdonságablakán lehetővé teszi, hogy az elemhez alkalmazásfüggő viselkedést rendeljünk. A lap felső részén láthatók a pillanatnyi elemhez tartozó jelzéskezelők. Az alsó részen található gombok segítségével az elem által kibocsátott jelzések közül lehet választani és a kezelője is létrehozható.
Új jelzéskezelő létrehozásához a *Signal entry* mező mellett található három pontra kell kattintani. Ekkor megjelenik a *Select Signal* párbeszédablak, amely felsorolja az elem által kibocsátható összes jelzést. A jelzések azok szerint a Gtk+ elemosztályok szerint vannak csoportosítva, amelyekben meghatározták őket.
A jelzés kijelölése után kattintsunk a Select Signal párbeszédablakban az *OK* gombra: a kijelölt jelzés neve megjelenik a Signals lapon a Signal entry mezőben. A Glade önműködően kitölti a *Handler* mezőt is, az `"on_<widget>_<signal>"` elnevezést használva. Ha igényeinknek ez nem felel meg, kézzel megváltoztatható.
A Signals lap alján további beviteli mezőket találunk, ahol alkalmazásfüggő adatok adhatók meg: a jelzést elfogó objektum stb. Ezeket mindig hagyjuk üresen, ugyanis szükségtelenek, amikor gnome-pythonnal dolgozunk.

Glade-projektfájlok

A Glade a projekt adatait a .glade kiterjesztésű XML fájlba menti. A Glade XML használata könnyűvé teszi különféle a projektfájlon dolgozó segédprogramok kifejlesztését, például új programnyelvekhez való kódlétrehozó írását.
Az új projekt első mentésekor a Glade felhozza a *Project Options* párbeszédablakot. A Project Options párbeszédablak legtöbb beállítása akkor és csak akkor számít, ha a Glade-et a projekt forráskódjának létrehozására használjuk. Néhány beállítás azonban – például a projekt könyvtára – akkor is fontos, ha a Glade-et csak az elrendezés megtervezésére használjuk.
Alapértelmezés szerint a Glade feltételezi, hogy az új projektet bejelentkezési könyvtárunk alá, a *Projects/project1* könyvtárba szeretnénk menteni. Ez nem feltétlenül egyezik meg az akaratunkkal. Én általában abba a könyvtárba szoktam menteni, ahonnan a Glade-et indítottam. Szerencsére könnyű megváltoztatni a projekt könyvtárát. Kattintsunk a *Browse…* gombra a *Project Directory* beviteli mező mellett, amely



3. kép A Project Options párbeszédablak

után a *Select the Project Directory* párbeszédablak jelenik meg. Ez a párbeszédablak a Glade aktuális munkakönyvtárát választja ki alapértelmezettként, ezért csak az OK gombot kell megnyomni. Ezután a Project Directory mező a Project Options párbeszédablakban a pillanatnyi munkakönyvtárat fogja mutatni, és a *Project Name* mező üres lesz. Írjuk be az új projektnevet, ennek hatására a Project Name és a *Project File* mezők frissülnek (lásd *3. kép*). Az OK lenyomása után a projekt a megadott projektfájlba íródik.

A libglade használata

Ha egyszer létrehoztunk egy Glade-projektfájlt, grafikus felületet (amelyet a projektfájl ír le) hozhatunk létre a gnome-python libglade modulját használva, és programból elérhetjük a szerkezet elemeit.

```
import libglade
loader = libglade.GladeXML
       ➥("helloworld.glade", "window1")
```

A libglade programkönyvtár megadja a GladeXML-osztályt, ez végzi a munka nagy részét. Egy elemszerkezet betöltéséhez a GladeXML-osztály egy példányát kell létrehozni, átadni neki a Glade-projektfájl nevét és a legfelső elem nevét, amelyet meg akarunk jeleníteni. Jegyezzük meg, hogy a szerkezet bármelyik elemét megadhatjuk itt, még akkor is, ha egy legfelső szintű ablakban mélyen el van ásva. Ez lehetővé teszi, hogy az összetett szerkezeteket – például egy bonyolult, füzetszerű felület lapjait – több Glade-projektfájlba osszuk el. Könnyen kezelhetők a dinamikus látványtartalmú projektek is, csak azt az összetevőt kell betölteni, amelyik éppen szükséges. Ha az elemszerkezet már be van töltve, egy adott elemet a GladeXML `get_widget` eljárásával kaphatunk meg. A `get_widget` a kért elemet adja vissza; ha pedig az elem nem található, akkor *None*-t.

```
window1 = loader.get_widget("window1")
if window1:
    window1.set_title("Szia Világ!")
```

Jelzéskezelők hozzákapcsolása

A GladeXML leghatékonyabb jellemzőinek egyike, hogy képes összekapcsolni a Python által hívható (Python callable) objektumait (eljárásokat, függvényeket stb.) a Glade-projektfájlban megnevezett jelzéskezelőkkel. Mindezt a `signal_autoconnect` eljárás teszi lehetővé.

A signal_autoconnect csak egy értéket vár: egy olyan szótárt, amely a jelzéskezelők nevét Python által hívhatókra képezi le. Minden egyes jelzéskezelőre, amelyet a Glade-projektfájlban megadtunk, a signal_autoconnect a meghatározott szótárból kikeresi a megfelelő Python által hívhatót, s ha egyező bejegyzést talál, akkor a jelzéshez kapcsolja – azaz a Python által hívható válik a jelzés kezelőjévé.

```
def button1_click_handler(*args):
    print "Ne nyomd meg a gombot!"

signal_handlers = {
    # Kilépés a főhurokból, ha a felhasználó
    # bezárja a főablakot.
    'on_window1_delete_event': gtk.mainquit,
    # A button1_click_handler hívása, ha a
    # felhasználó megnyomja a button1 gombot.
    'on_button1_clicked': button1_click_handler
    }

loader.signal_autoconnect(signal_handlers)
```

GladeBase

A libglade nagymértékben csökkenti a kézi kódolás szükségességét gnome-python alkalmazások esetében. Az elemszerkezeteket megtervezhetjük Glade-del, és két-három sor kód segítségével be is tölthetjük. Mindez több száz sort venne igénybe, ha közvetlen *pygtk* hívásokat használnánk. Ráadásul a viselkedés is egyszerűen hozzáadható a Python által hívhatók szótárának összeállításával és a `GladeXML.signal_autoconnect` átadásával, ahelyett hogy állandóan az elemcsatoló eljárásokat kellene hívogatni. A libglade sok munkát takarít meg, ennél azonban többre is képes, például a nagyméretű Python-alkalmazások gyakran egy kis főprogramból és valahol a Python elérési útjában elhelyezkedő kapcsolt csomagokból állnak. A karbantartási költség csökkenthető lenne, ha az alkalmazás Glade-projektfájljai a Python csomagjaival egy helyen lennének, és futás közben relatív útvonalneveken keresztül importálhatnánk azokat. Az is jó lenne, ha az elemeket közvetlenül valamilyen felhasználói felület objektumpéldányán keresztül érhetnénk el, és nem a `GladeXML.get_widget` segítségével kellene megkeresni őket. Végül hasznos volna, ha az objektum névterében található hívhatók szótárának összeállítását a gépre lehetne bízni, és az eredményt át lehetne adni a `signal_autoconnect`-nek. Ez lehetővé tenné az ügyfelek számára, hogy a jelzéskezelőket az objektum eljárásaiként adják meg, és a kezelőket ne közvetlenül kelljen bejegyezni.
A következő fejezet a GladeBase-modult írja le, amely rendelkezik ezekkel a tulajdonságokkal. A GladeBase a libglade szolgáltatásait tartalmazza, úgy módosítva, hogy azok az MVC (model view controller) tervezőmintához illeszkedjenek (lásd az 1. listát a 15. CD Magazin/Glade könyvtárban). A GladeBase két alapvető exporttal rendelkezik: az `UI` és a `Controller` osztállyal.

GladeBase.UI

A `GladeBase.UI` az MVC tervezőminta *View* összetevőjének felel meg. Ez felelős az elemszerkezetnek a Glade-projektfájl alapján való létrehozásáért, valamint az alkalmazás látható tartalmának frissítéséért egy hozzárendelt vezérlő irányítása alatt. A GladeBase.UI a libglade GladeXML-osztályának leszármazottja, így minden korábban tárgyalt eljárást örököl.
A GladeBase.UI létrehozója (konstruktora) három értéket vesz át: a Glade-projektfájl nevét, amelyből az elemszerkezetet betölti; a gyökérelemként szereplő elem nevét, és egy választható kulcsszót, a `gladeDir`-t. Ez annak a könyvtárnak a relatív útvonalnevét tartalmazza, ahol a Glade-projektfájlokat kell keresni.

2. lista PathFinder.py

```
#!/usr/bin/env python
"""PathFinder.py
Ez a modul feloldja a Python útvonalhoz képest
       relatív útvonalakat."""

import sys, os

class Error(Exception):  """Akkor jön elő, ha
       az útvonalnév nem feloldható"""

def find(pathname):
    """Egy hely útvonalnevét feloldja a
       Python útvonalon."""

    if os.path.isabs(pathname):
        return pathname
    for dirname in sys.path:
        candidate = os.path.join(dirname,
                    pathname)
        if os.path.isfile(candidate):
            return candidate
    raise Error("%s nem található a Python
                útvonalon."
                  % `pathname`)

def main():
    """A modul fősora (egyedülálló végrehajtás
                       esetén)"""
    def testFind(path, expectError=0):
        """Megpróbálja feloldani az útvonalat,
diagnosztikai üzenetekkel."""
        print "%s feloldása..." % path
        try:
            print find(path)
        except Error, info:
            if expectError:
                print "Ahogy vártuk:", info
            else:
                print "HIBA:", info

    # Próbáljunk valami ismert dolgot
    # megtalálni az útvonalon,
    # relatív és abszolút útvonalakat egyaránt
    # használva.
    testFind("socket.py")
    join = os.path.join
    socketPath = join(sys.prefix, join("lib",
        join("python", "socket.py")))
    testFind(socketPath)

    # Próbáljunk egy valószínűleg nem létező
    # dolgot
    # megtalálni.
    testFind("I_Dont_Exist.nonexistent_path", 1)

if __name__ == "__main__":
    main()
```

A gladeDir kulcsszó alapértelmezett értéke a pillanatnyi könyvtár. A fájlnévvel összekapcsolva a Glade-projektfájl relatív elérési útját kapjuk. Furcsának tűnhet a gladeDir és a fájlnév együttes használata, ahelyett hogy a Glade-projektfájl nevét egyszerűen relatív elérési útjával adnánk meg. Ez a szétválasztás azonban csökkentheti az olyan alkalmazások karbantartási költségeit, amelyek egyetlen alcsomagban tartják a Glade-projektfájljait. Az ilyen alkalmazások a GladeBase.UI olyan alosztályát adhatják meg, amelybe a gladeDir értéke be van drótozva.

```
import GladeBase

class UIBase(GladeBase.UI):
  def __init__(self, filename, rootname):
    GladeBase.UI.__init__(self, filename,
      ➥rootname, gladeDir="MyApp/GladeFiles")

class MainWinUI(UIBase):
  def __init__(self):
    UIBase.__init__(self, "main_win.glade",
      ➥"window1")
```

Ezután az alkalmazás minden UI-osztályát ebből az alosztályból származtathatja. Ilyen módon az alkalmazás egy helyen adhatja meg az összes Glade-projektfájl könyvtárának relatív útvonalát.
A PathFinder.py segédmodul lehetővé teszi a GladeBase.UI számára, hogy a Python-útvonalon keressen fájlokat. A `PathFinder.find` függvény egyetlen argumentuma az elérési út. Ha az útvonalnév abszolút, további feldolgozás nélkül visszatér. Ha relatív, akkor a `find` függvénysorban minden egyes Python útvonalbejegyzéssel összekapcsolja, így áll elő a megvizsgálandó útvonalnév. Ha ez létezik, a függvény visszatér. Ha egyik ilyen útvonalnév sem létezik, akkor `PathFinder.Error` kivétel lép fel (lásd a *2. listát*).
A `GladeBase.UI.__getattr__` eljárás teszi lehetővé az ügyfelek számára, hogy a GladeBase.UI-szerkezetben elhelyezkedő elemekhez úgy férjenek hozzá, mintha azok a példány tulajdonságai lennének. A `__getattr__` eljárás feltételezi, hogy a hívó által megadott tulajdonságnév az elem neve, és az elemet a GladeXML.get_widget segítségével kikeresi. Ha az elemet megtalálta, a gyorstárba új példányváltozóként kerül be, ezáltal a későbbiek folyamán gyorsabban lehet elérni. Ha a keresett elem nem található, `__getattr__` `AttributeError` hibajelzést ad vissza. Ha az elemszerkezet egynél több azonos nevű elemet tartalmaz, akkor nem lehet megmondani, hogy a GladeBase.UI melyiket adja vissza. A GladeBase.UI használata során jól tesszük, ha az elemeket úgy nevezzük el, ahogy a Python-példánytulajdonságokat: az objektumon belül minden név legyen egyedi, és érvényes Python-azonosítóval rendelkezzen.
Az alkalmazásfüggő UI-osztályok a GladeBase.UI-osztályt általában olyan eljárásokkal bővítik ki, amelyek összetett frissítéseket végeznek a felhasználói felületen.

GladeBase.Controller

A GladeBase.Controller az MVC *Controller* összetevőjének felel meg. A Controller úgy válaszol a felhasználói bemenet eseményeire, hogy azokat az alkalmazás belső adatmodelljének állapotváltozásaira fordítja le. Hasonlóképpen, az adatmodell állapotváltozásai esetén frissíti a felhasználói felületet.

A GladeBase.Controller nem segít válaszolni az alkalmazás adatmodelljében bekövetkezett változásokra, de a jelzéskezelő eljárásokat önműködően összeköti a Glade-projektfájlban megadott jelzéskezelőkkel. A GladeBase.Controller konstruktora egy értéket vesz át, ez a GladeBase.UI egy példánya és a vezérelendő felhasználói felület. A betöltés során az új GladeBase.Controller-példány átfut (traverses) az osztály szerkezetén, és a példány névterében található összes hívható objektumból felépíti a szótárt (az átfutás – traversal – a példány szótárával kezdődik, amennyiben példánytulajdonságként hívhatókat adtak meg).
Ezután a GladeBase.Controller ezt a szótárt a megadott GladeBase.UI-példány signal_autoconnect eljárásának adja át. Az alkalmazásfüggő vezérlőosztályok a GladeBase.Controller-osztályt terjesztik ki a jelzéskezelő eljárások megadásával.

```
class Controller(GladeBase.Controller):

    def __init__(self, ui):
        ...
        GladeBase.Controller.__init__(self, ui)

    def on_window1_delete_event(self, *args):
        gtk.mainquit()

    def on_button1_clicked(self, *args):
        print "1. gombot megnyomták."
```

Vezérlővázak létrehozása

A GladeBase önműködővé teszi a Gtk+ elemszerkezetek Python objektumszerkezetekké alakítását, és önműködően összekapcsolja a Python jelzéskezelőivel, de még mindig megkívánja, hogy azonosítsuk és megvalósítsuk az összes Glade-projektfájlban megadott jelzéskezelőt. A tiszta Gtk+ projekteknél ez nem gond, hiszen csak azok a jelzéskezelők léteznek, amelyeket kifejezetten megadtunk. Ellenben, ha a Glade-et Gnome-alkalmazás készítésére használjuk, sok jelzéskezelő önműködően megjelenik. Például egy új Gnome-alkalmazásablak szabványos menüsorral jön létre, és ezek a menüelemek előre megadott jelzéskezelőket tartalmaznak. Nagyon unalmas lenne egy Gnome-alapú projektet átböngészni, az előre megadott jelzéskezelőket kézzel megkeresni, és ezeket az alkalmazás vezérlőihez hozzáadni. Ahogy korábban megjegyeztük, a Glade-projektfájlok XML-formátumban vannak (pillanatnyilag nincs még DTD, amely leírná a projektfájl szerkezetét, de e nélkül is könnyű megérteni). A Python 2.0 tartalmaz egy XML-programkönyvtárat, amely *James Clark* Expat könyvtára fölé rétegződik. Ezért meglehetősen egyszerű olyan Python-alkalmazást írni, amely végigszalad a Glade-projektfájlon, azonosítja az adott elemszerkezetekben hivatkozott jelzéskezelőket, és a Controller-modul vázát ehhez a szerkezethez elkészíti.
A `GladeProjectSignals.py` (a 3. listát lásd a 15. CD Magazin/Glade könyvtárban) a Glade-projektfájlból gyűjti ki a jelzéskezelők adatait. A `WidgetTreeSignals` osztály felépíti az elemszerkezetet képviselő `XML DOM` (Document Object Model) -fát és minden, a jelzéskezelőkre vonatkozó hivatkozást feljegyez. A GladeProjectSignals-osztály betölti a Glade-projektfájlt, és a WidgetTreeSignal példányokból az összes legfelső szintű elemhez felépít egy szótárat.
A WidgetTreeSignals létrehozójának argumentuma egy DOM-csomópont. Feltételezi, hogy ez a csomópont egy elemet ír le, és elvárja, hogy az elem nevét megadó névcsomópontot tartalmazza. Ha ezek a feltételek teljesülnek, a WidgetTreeSignals feljegyzi a csomópont kezelőgyermekének értékét, amely nem más, mint a jelzéskezelő neve. Egyébként a WidgetTreeSignals feltételezi, hogy a csomópont gyermekcsomópontokat tartalmaz, és ezekkel folytatja az olvasást.
A `GladeProjectSignals` hasonlóképpen egyszerű. A Python a Glade-projektfájlt `xml.dom.minidom` csomagjának segítségével betölti a DOM-fába. Ezután a fában megkeresi a legfelső szintű elemeket (a Glade-tervezőfájl más legfelső szintű csomópontokat is tartalmaz, például a GTK-felület és projekt csomópontjait).
A GladeProjectSignals minden megtalált elemcsomóponthoz egy új WidgetTreeSignals-példányt hoz létre, amely az elem és leszármazottai által meghatározott jelzéskezelőket sorolja fel. Minden egyes WidgetTreeSignal-példány a legfelső szintű elem nevével megjelölt *self.widgets* nevű szótárba kerül.
Ha a `ControllerGenerator.py`-nek (lásd a 4. listát lásd a 15. CD Magazin/Glade könyvtárban) átadunk egy Glade-projektfájlnevet és a fájlban megadott egyik legfelső szintű elem nevét, akkor kinyomtatja az elemhez és gyermekeihez tartozó Controller vázát. A munka oroszlánrészét a `ControllerGenerator` osztály végzi. Ez az osztály tartalmazza a `generate` eljárást, amely a Glade-projektfájl nevét és a legfelső szintű elem nevét veszi át. A generate eljárás a szóban forgó elem jelzéskezelőihez a GladeProjectSignals egy példányán keresztül jut hozzá. Ezt követően elkészíti ezeknek a kezelőknek a vázát. Egy mintakarakterlánc és a Python karakterlánc-formázó operátorainak segítségével a generate létrehozza a Controller modul vázát alkotó karakterláncot, és visszaadja azt a hívónak.

Összegzés

A Glade, a libglade és a gnome-python nagymértékben képes csökkenteni a Python nyelven írt Gtk+ és a Gnome-alkalmazások fejlesztéséhez szükséges munkát. A cikkben bemutatott eszközök a Glade-elemszerkezetek Python objektumszerkezetekké történő alakításával, a vezérlőkben megadott jelzéskezelők összekapcsolásával és a vezérlővázak létrehozásával a karbantartás költségét tovább csökkentik.

Ajánlott irodalom

Design Patterns: Elements of Reusable Object-Oriented Software, írta Gamma, Helm, Johnson és Vlissides (Addison-Wesley, 1994). Leírja az Observer mintát, idézi a Smalltalk Model-View-Controller keretrendszerét mint a minta egy korai példáját.

Mitch Chapman (chapman@bioreason.com) vezető programmérnökként dolgozik a Bioreasonnél. Az új-mexikói Santa Fében él, ahol csaknem egyidejűleg hódolhat a Python-programozás, a hódeszkázás, a sziklamászás, valamint a Napba nézés és a repülés örömeinek.

Kapcsolódó címek

A Glade projekt honlapja
➲ http://glade.pn.org/
A PyGTK burkolókönyvtár honlapja
➲ http://www.daa.com.au/~james/pygtk
A libglade programozói leírása
➲ http://developer.gnome.org/doc/API/libglade/libglade.html
A Gtk+ projekt honlapja ➲ http://www.gtk.org/
Glade tananyag
➲ http://linuxfocus.org/English/July2000/article160.shtml