Ximba Radio: Grafikus felület fejlesztése az XM Satellite Radio programhoz GTK+/Glade segítségével

A Glade-ről azt mondják, hogy segítségével a grafikus programok prototípus-készítése egyszerű és gyors folyamattá tehető. Hogy a saját problémámnál, a számítógépemen lévő XM Satellite Radio programnál maradjak, úgy döntöttem, megvizsgálom, hogy pontosan mennyire egyszerű és gyors.

Ahogy az amerikai televíziózás egyre inkább elsüllyed a kitalált valóság feneketlen mélységeiben, egyre inkább azt veszem észre magamon, hogy kezdek visszatérni az elektronikus szórakoztatás gyökereihez, a rádiózáshoz. Ennek a médiumnak a legfrissebb megtestesülése a műholdas rádiózás, amely a csatornák széles választékának elérhetőségét biztosítja bárhonnan, ahova autóval el lehet jutni.
Mivel több időt töltök monitor előtt, mint az autóm kormánykerekénél, örömmel fedeztem fel a műholdas rádió egy számítógépes megoldását. Az ***XMPCR*** az ***XM Radio*** műholdas rendszer számára készült USB-csatlakozós vevőegysége, amelyet elsősorban Microsoft Windows rendszerekhez árusítanak. Linux alatt az eszközt az ***OpenXM*** projekt támogatja, ez egy Perl parancsfájlokból álló gyűjtemény, amely az eszköz vezérlését hálózati démonként működve oldja meg. Sajnos a démon egyetlen felhasználói felülete egy korlátozott szöveg alapú eszköz.

A projekt eredményeinek előzetes áttekintése

A ***Ximba Radio*** az ***OpenXM***-hez készült grafikus felületként született. A program egy végletesen egyszerű főablakot biztosít az aktuális csatorna információival, amely a csatornalistával és a felhasználó kedvenceivel bővíthető. Az ablak felső részén végighúzódó gombsáv biztosítja a rádió könnyű kezelését, az állomások közti mozgást, valamint a programbeállítási lehetőségek gyors elérését. A menüsor megadja a felhasználók számára egy hagyományos asztali program kényelmét.
A csatornalista többféle formátumban is megjeleníthető. A fő csatornalista ablaka minden csatornát megmutat, míg a kategóriák szerinti füleken csak a megfelelő csatornák szerepelnek. Külön füleken keresztül érhetőek el a felhasználó által kijelölt előadók és kedvenc csatornák, valamint a pillanatnyi folyamatelőzmény-lista. A kategóriafülek a ***Preferences (Beállítások)*** párbeszédablakban elrejthetőek, s ugyanitt tudjuk beállítani az előadásra és a kedvencekre vonatkozó megjegyzéseinket.

1. kép A főablak, a csatornalisták és a beállítások. A második változat az egyszerűsített formátumot mutatja a csatornalista elrejtésével.

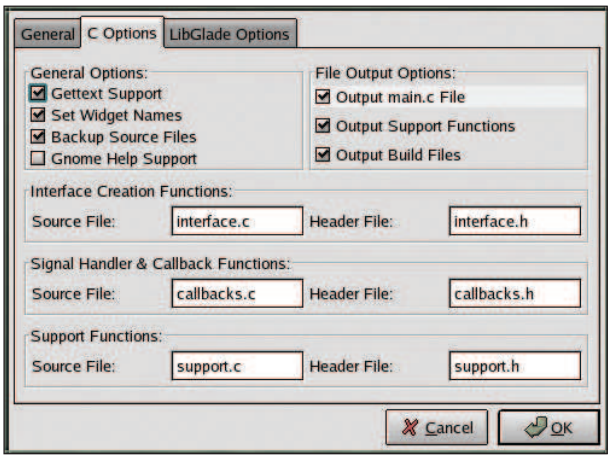
A ***Ximba Radio*** hátterét az ***OpenXM*** démon jelenti. Ez a ***Perl***-parancsfájl és a kapcsolódó ***Perl*** modul vezérli az USB kapun létrejövő kapcsolatot. A démon a beállításait a beállítófájlból vagy parancssori paramétereken keresztül kaphatja meg. A démonnal egy TCP-kapun keresztül tarthatjuk a kapcsolatot, amelyhez megadható az elfogadható ügyfelek listája. A démon futtatható Windows rendszerek alatt is.

Tervezési célok

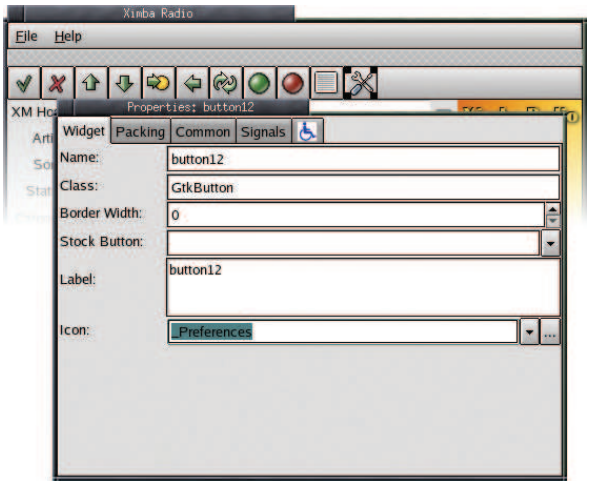
Célom, hogy a ***Windows*** alatti változathoz hasonló képességekkel rendelkező, minél egyszerűbb felülettel bíró, könnyen beállítható programot hozzak létre. További feltétel volt, hogy a megvalósítás ne tartson tovább egy munkahétnél (40 óra).
Törekedtem arra is, hogy a felhasználói felület a lehető legnagyobb mértékben független maradjon a program törzskódjától. Egy program kódjának alkalmasnak kell lennie bármilyen megfelelő felhasználói felülettel való használatra, így jó tervezés esetén kevés munkával egy Webes felülettel való együttműködésre is könnyen rávé-

© Kiskapu Kft. Minden jog fenntartva

© Kiskapu Kft. Minden jog fenntartva



2. kép A Glade Options ablaka lehetővé teszi a kód előállítását, fájlnevek megadását és a nemzetközi szövegek támogatásának kiválasztását



3. kép A Glade toolbar-eszköze, és a tulajdonságok, amelyekkel ikonok és gombok hozhatók létre az eszközsávon belül

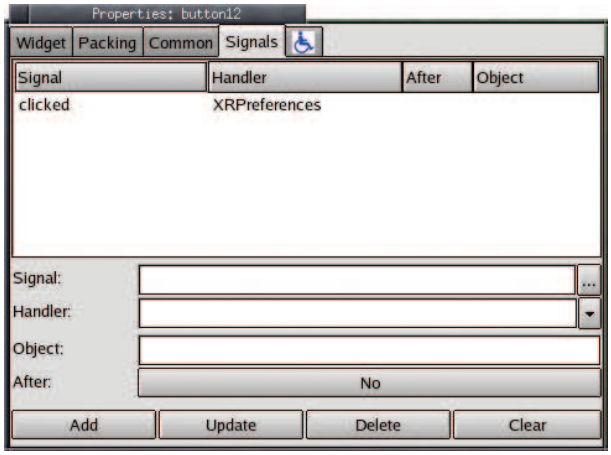
hetőnek kell lennie. Ez a cél összhangban áll a *GNOME* fejlesztői útmutatóival és a *Glade* jövőbeli terveivel egyaránt.
Ezeknek a céloknak megfelelően úgy terveztem, hogy egyetlen C fejállományt és egyetlen C modult fogok használni. A fejlesztés felgyorsítása és néhány időrabló probléma megoldása érdekében globális változók használata mellett döntöttem. Ne feledjük, hogy egy egyszerű asztali alkalmazásról és nem valami üzleti célú 24/7 hibatűréssel rendelkező programóriásról van szó. Ha jól kezeljük a kérdést, a globális változók használata a jövőbeli frissítéseknél kiküszöbölhető.

Ismerkedés a Glade-del

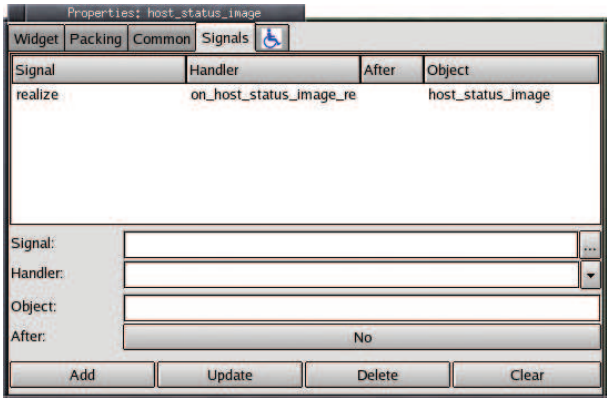
A célok meghatározása után belevágtam a *Glade*-del a felhasználói felület megformálásába – a következő részben részletezem ennek a folyamatnak a kóddal kapcsolatos vonatkozásait. Úgy állítottam be a *Glade*-et, hogy C-kódot állítson elő, az *Options* (lehetőségek) párbeszédablakban minden egyéb kérdésében elfogadtam az alapértelmezett beállítást. Itt a legfontosabbak a forrás- és fejállományok, amelyek a felhasználói felület meghatározásait tartalmazzák (*interface.c*), az eszköz-visszahívások (*callbacks.c*), és a *main.c* fájl előállításának beállításai.
A *Glade* a programok írására egy teljesen felépített környezetet hoz létre, amely tartalmazza a forráskönyvtárat (*src*) és a képfájlok tárolására szolgáló könyvtárat (*pixmaps*) is. A *GtkImage* eszközök által használt képeknek *xpm* formátumúnak kell lenniük. Az egyéb létrehozott fájlok között találjuk az *autoconf* és *automake* sablonokat és a *pot*-fájlokat a nemzetközi karakterláncok tárolására.
A nemzetközi támogatás nem kötelező, ezt a GNU *gettext* támogatása kezeli. Például az *interface.c* fájlban találhatunk *gettext* által engedélyezett karakterláncokat. Még ennek a lehetőségnek az engedélyezése esetén sem kell magunknak létrehozni a *pot*-fájlokat egyik nyelvhez sem, a Glade képes bármilyen szöveglánc használatára, amit alapértelmezett nyelvként adunk meg.
Tapasztalatom szerint a *Ximba Radio* prototípusának *Glade*-del történt elkészítése során a létrehozott fájlok közül mindössze kettő – a *main.c* és a *callbacks.c* – kézzel való módosítására volt szükség. Az előbbiben csak néhány kisebb kiegészítésre volt szükség a program beállításainak kezelésével kapcsolatos előzetesi választási lehetőségek terén. A *callbacks.c* fájl módosításainak zömét a C modulom, a *utils.c* felé történő hívásátadásai jelentették.
A projekt *Glade*-ben történő módosításai és a C-kód újbóli előállítása során a *callbacks.c* fájl új függvényekkel egészült ki. Szerencsére a *Glade* jól tudja, hogy mikor létezik már egy hívás, és egyszer sem írta felül a módosításaimat. Az viszont sajnos néha előfordult, hogy egy már létező függvényt újra hozzáfűzött. Időről-időre szükség volt ezeknek a függvényeknek a kézzel történő kitörlésére.
Ha *libGlade*-et használunk, amely a C-kód létrehozása helyett közvetlenül a Glade XML fájlt dolgozza fel, nem létezik ez a probléma, de a *libGlade* tárgyalása már meghaladja e cikk kereteit.

A felhasználói felület fejlesztése a Glade-del

A *Ximba Radio* két elsődleges ablakot igényelt, a *Főablakot* (*Main Window*) és a *Beállítások párbeszédablakot* (*Preferences Dialog*), valamint számos másodlagos előugró ablakot. A főablak gombsorát a *Glade* eszközsáv-eszközével hoztam létre, ehhez adtam kézzel a gombokat. A *GTK+* gombjai szöveget vagy képet tartalmazhatnak. A *Glade*-ben választhatunk alkalmazásképek (application images), beépített gombok (stock buttons) és beépített ikonok (stock icons) közül. A beépített gombok ugyanazokat az ikonokat használják, mint a beépített ikonok azzal, a különbséggel, hogy ezeknél az gyorsúgó nem elérhető. Emiatt én inkább az ikonok használatát javaslom, a stock button mezőt pedig hagyjuk üresen.
Az eszközsáv minden gombja egyetlen meghívható függvénnyel rendelkezik, amely a kattintáshoz (*click*) kapcsolódik. A hívandó függvénynek bármilyen nevet adhatunk, és igény szerint paraméterként magának az eszköznek a nevét is átadhatjuk. A kattintáshoz kapcsolódó függvények esetében ez utóbbi nem szükséges. A *realize* eseményekhez kapcsolódó függvényhívásokban – amiről hamarosan szó is esik – az eszköz nevét is átadjuk a függvénynek.
Három *GtkImage* eszközt helyeztem el a főablakon. Az első egy *State* (*Állapot*) ikon, amelyet a *Host name* (*Gépnév*) me-



4. kép Az eszközsávon lévő gombokhoz egyetlen függvényhívás rendelhető, amely a rá való kattintásra (clicked) lép működésbe



5. kép A State (állapot) ikonnak egy realize-függvényhívásra is szüksége van ahhoz, hogy megkapjuk az eszköz nevét és lehetővé váljon annak futási időben történő frissítése

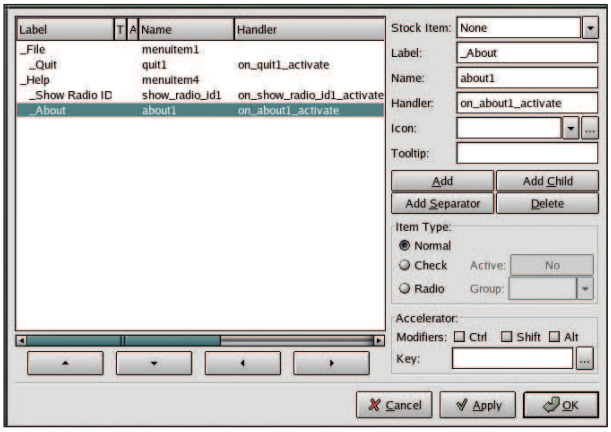
© Kiskapu Kft. Minden jog fenntartva

ző jobb felére raktam. A ***Remove*** (***Eltávolítás***) ikont állítottam be arra, hogy mutassa, ha nincs kapcsolat a démonnal, (a *Glade* egyébiránt, rengeteg beépített ikont tartalmaz).
A csatlakoztatott állapot feltüntetésére az ***Apply*** (***Alkalmaz***) ikont használtam. Az ikonok futási időben történő megváltoztatásához ennek a *GtkImage*-nek az eszközazonosítóját egy ***realize*** függvényhívásba mentettem. A használat közben ez az ikon az ***Off*** (***Kikapcsolt***) ikonra is beállítható, amely a csatlakoztatott de elnémított állapotot jelzi. A következő szakaszban meg fogom vizsgálni ezekhez a változtatásokhoz kötődő forráskódot is.
A ***Favourites*** (***Kedvencek***) gombok ábrája egy plusz jel. A *Glade* és a *GTK+* ezeket Add (hozzáadás) ikonoknak nevezi. Ezek a gombok egyetlen függvényhívással rendelkeznek, amelyet a hozzájuk kapcsolódó ***click*** (***kattintás***) esemény vált ki. A függvény az aktuális előadót vagy csatornát hozzáadja a megfelelő kedvencek listájához. A főablak tetején látható menüsort a *Glade* beépített ***Menüszerkesztőjével*** (***Menu Editor***) hoztam létre. A szerkesztőnek rengeteg beállítási lehetősége van, ezek közül én csak a ***Label*** (***Címke***), ***Name*** (***Név***) és ***Handler*** (***Kezelő***) tulajdonságokat használtam, ez utóbbit a menüpont kiválasztásakor hívandó függvény megadására.
Egy ***Notebook*** (***Jegyzetfüzet***) eszköz biztosítja a teljes csatornalista elérését csakúgy, mint a kategóriák, kedvencek és folyamatok szerinti listákét. Ezek kezelése a ***CList*** eszközön keresztül biztosított. A *Glade* teljes mértékben támogatja ezt az eszközt annak ellenére, hogy a *GTK+* az új kódokban már az újabb és bonyolultabb ***Tree and List*** (***Fa és Lista***) eszközt részesíti előnyben. Ezt a vitatható döntést kicsit később részletezni fogom majd.

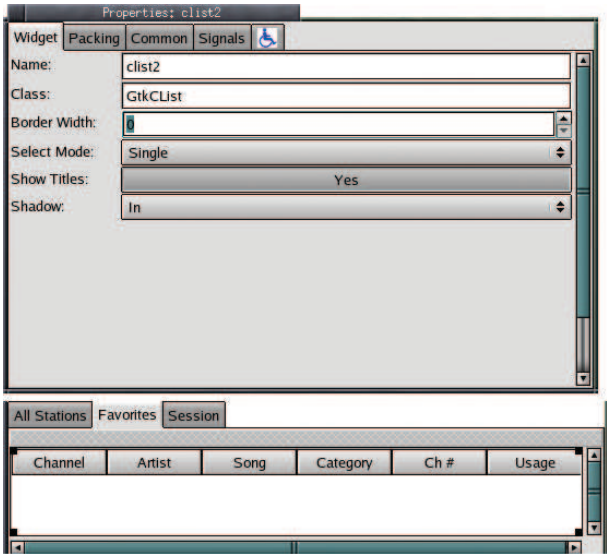
A kódolás folyamata

A *Glade* a függvényhívásokhoz üres függvényeket hoz létre, amelyeket gyakran ***csonkoknak*** (***stub functions***) is neveznek. A csonkok lehetővé teszik, hogy a prototípus fejlesztése során egy egyszerű folyamat szerint haladjunk: megtervezzük a felhasználói felületet, létrehozzuk a kódot, megírjuk a függvényhívásokat, teszteljünk és ismételjünk. A hívások kódolásának nagy részét – a menüből való kilépés függvényeitől eltekintve – a felhasználói felület befejezése után végeztem el. Később tértem vissza a függvényhívások feltöltéséhez. Ez a módszer lehetővé tette, hogy a program elrendezésével kísérletezgessek anélkül, hogy túlságosan mélyen bele kelljen bonyolódnom az egyes részek működésébe. A módszer része annak a korábban említett törekvésnek is, hogy a felhasználói felület kódja minél inkább elkülönüljön a program forráskódjától. A két rész szétválasztásával lehetővé teszem a felhasználói felület jövőbeli módosítását anélkül, hogy ez komolyabban érintené az alapvető forráskódot. A felhasználói felület a függvényhívásokon keresztül kapcsolódik a programkódhoz, hiszen a felület eseményeit ezek képezik le a program szintjére, ahol ennek hatására végrehajtódik valamilyen tevékenység.
A függvényhívások különböző felületekkel rendelkeznek. A gombok ***click*** (***kattintás***) eseményei olyan hívásokat igényelnek, amelyek bemeneti paraméterként tartalmazzák a gomb-eszköz azonosítóját és a felhasználó adatait. A ***CList select-row*** (***sorkiválasztás***) eseményéhez tartozó hívás, amely egy sorra való kattintáskor következik be, öt paramétert vár. Ha hagyjuk, hogy a *Glade* hozza létre ezeket a függvényeket, hamar megismerhetjük ezeket a különböző felületeket. Valójában, mivel az API-hívások nem túl jól dokumentáltak – vagy legalábbis nem könnyű megtalálni a leírásokat – a parancsformátum megismerésének legjobb módja ha hagyjuk, hogy a *Glade* hozza létre a függvényeket.
A hívások kódjának kitöltését végezhetjük közvetlenül a ***callbacks.c*** állományban, de ezt a fájlt a jövőben, a ***lobGlade***-re való átálláskor már nem fogom használni, ezért ehelyett a paramétereket rendszerint egyenesen a utils.c fájlban lévő hasonló függvénynek adom át, amely a tényleges munkát végzi. Ezt az általános szabályt egy fontos kódrészlettel szegtem meg: az eszközazonosítók globális változókhoz való rendelését végző kód a callbacks.c fájlba került. Az 1. listán láthatjuk, hogyan használom egy hívásban a globális változót a ***Preferences*** (***beállítások***) párbeszédablak azonosítójának mentésére.
Több okból is szükségessé vált az egyes eszközök nyomonkövetése. Az első, hogy némelyik ikon a program állapotaitól függően dinamikusan változik. A második, hogy sok ablakot csak ideiglenesen kell megjeleníteni, ezek állandó

© Kiskapu Kft. Minden jog fenntartva



6. kép A Glade menüszerkesztője rengeteg lehetőséget kínál, de a prototípus elkészítéséhez ezek közül csak a Label, Name és Handler beállítására van szükség



7. kép A Glade teljes mértékben támogatja a CList eszközt, amely az egyszerű listák kezelésére alkalmasabb, mint a Tree and List eszköz

létrehozása és megsemmisítése felesleges lenne, sokkal kényelmesebb ezeket egyszer létrehozni és szükség szerint egyszerűen megjeleníteni vagy elrejteni. Végül pedig a *Glade* által létrehozott ***CList*** frissítésének futásidőben kell történnie. Az eszközazonosítókat tartalmazó változók érvényessége csak az ***interface.c*** fájlra terjed ki, ami azt jelenti, hogy azok a függvények, amelyek ezen a *Glade* által létrehozott fájlon kívül esnek, nem tudják egyszerűen módosítani ezeket az eszközöket.

A probléma megoldása végett minden olyan eszközhöz, amelyhez futásidőben is hozzá akartam férni, beállítottam egy felismerő eseményt. A *Glade* lehetővé teszi az ***interface.c*** fájlban definiált változók nevének megadását. Az említett eseményhez rendelt függvényhívás ennek a változónak az értékét kapja meg objektumparaméterként. A hívásban ez az érték egy olyan globális változóba kerül, amely az ***xr.h*** fájlban – a projekt számára általam létrehozott egyetlen fejállományban – került meghatározásra. Valamennyi globális változó érvényességét

1. lista A Preferences (beállítások) párbeszédablak az első kérés időpontjában jön létre, az eszközazonosító pedig globális változóba kerül

```
void
XRPreferences     (GtkButton       *button,
                   gpointer         user_data)
{

   /* Ha korábban még nem volt megnyitva,
    * hozzuk létre a párbeszédablakot.
    */

   if ( XR_Preferences_Window == NULL )
   {
      XR_Preferences_Window =
create_preferences();
      gtk_widget_realize(XR_Preferences_Window);
   }
   ...
}
```

2. lista A globális változók és függvények deklarációja az xr.h fájlban kap helyet

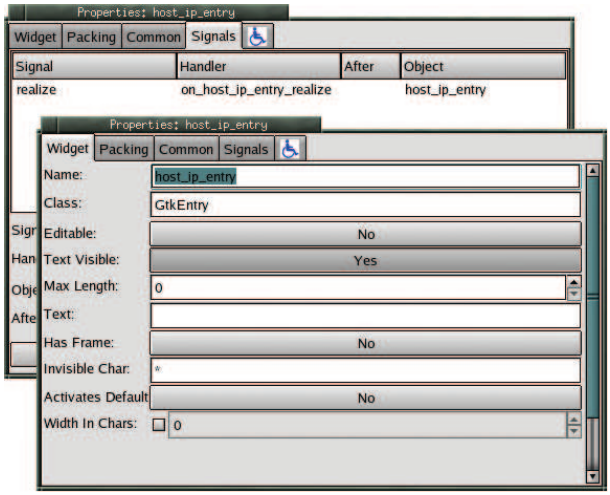
```
code:
#ifdef XR_CB_C
GtkWidget *XR_Msg_Window = NULL;
GtkWidget *XR_Msg = NULL;
void XRUMsg();
void XRUInit();
#else
extern GtkWidget *XR_Msg_Window;
extern GtkWidget *XR_Msg;
extern void XRUMsg();
extern void XRUInit();
#endif
```

3. lista A #define biztosítja a változók és függvények megfelelő elérhetőségét a C-modulokból

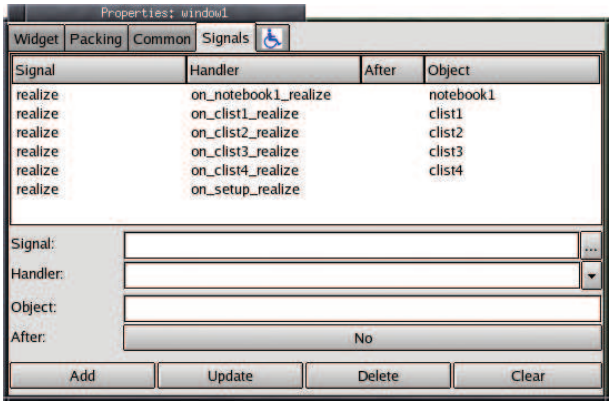
```
code:
#define XR_UTIL_C
#include <stdio.h>
#include <stdlib.h>
#include "xr.h"
...
```

a C-modul elején megadott ***#ifdef*** és ***#define*** előfeldolgozási direktívákkal szabályozzuk, amint az a 2. és 3. listákban is látható.

A módszer egyetlen problémája annak meghatározása, hogy mikor válik egy eszközazonosító elérhetővé. A ***realize*** eseményhez rendelt függvényhívás csak közvetlenül az



8. kép A realize események szerepe az eszközazonosítók egy függvényhívásnak való átadásában, amely azokat globális változókba menti



9. kép Közvetlenül a főablak megjelenítése előtt az eszközazonosítók és a CList ablakai többszörös függvényhívással globális változókba kerülnek

© Kiskapu Kft. Minden jog fenntartva

eszköz megjelenése előtt hajtódik végre, pedig néha már ez előtt is szükségünk van az eszközazonosítóra. Szerencsére létezik egy egyszerű megoldás. Az eszközt az ***interface.c*** állomány hozza létre még az eseménykezelők létrehozása előtt. Az eseménykezelő egy olyan függvény, amely a függvényhívást valamilyen eseménnyel kapcsolja össze. Ebből következően, mire a ***realize*** esemény beállítása megtörténik, már minden helyi változó érvényes értékkel rendelkezik. Ez lehetővé teszi azt, hogy egyetlen eszközhöz több olyan függvényhívást is létrehozzunk, amelyek mindegyike az adott eszköz ***realize*** eseményéhez kötődik és amelyek más eszközök eszközazonosítóit mentik el. Például a ***Ximba Radio main window (főablak)*** eszköze rendelkezik olyan ***realize***-hívásokkal, amelyek elmentik a ***Channel Listing*** (***Csatornalistázó***) ablak mind a négy előre meghatározott ***CList*** eszközének eszközazonosítóját. Erre szükség is van, hiszen kezdetben ezek a ***CList*** eszközök nem láthatóak, csak miután a főablak is azzá vált, viszont a frissítésüket azonnal meg kell kezdeni. Ha nem használnám a főablakot a ***CList*** eszközök azonosítóinak elmentésére, nem tudnám elkezdeni a csatornainformációkkal való frissítésüket sem mindaddig, amíg legalább egyszer meg nem jelentek a képernyőn.

Az eszközök dinamikus megváltoztatása szintén megkívánja az eszközazonosító mentését. Erre a ***Ximba Radio*** állapotát jelző ikon az egyik megfelelő példa. Az állapotjelző ikon megváltoztatásához szükség volt arra, hogy csak a ***GTK+*** ikonkészletéből származó ikont használjak, és elmentsem a ***Glade*** által létrehozott ***GtkImage*** eszköz azonosítóját. Amikor a felhasználó megváltoztatja a program állapotát – akár azzal, hogy megszünteti a kapcsolatot a démonnal, akár a némítás engedélyezésével vagy tiltásával – az állapotot jelző ikont egyszerűen egy ***GTK+*** függvényhívással megváltoztatjuk, amint az a 4. listán is látható. A ***GTK+*** beépített ikonjainak teljes készlete a hálózaton elérhető ***GTK+*** leírásában található meg.

Nem a globális változók kérdése a módszeremben az egyetlen kérdés, amelyet a tapasztaltabb fejlesztők vitathatnak. A másik a kifogásolt ***GTK+*** eszköz, a ***CList*** használata, amelyet oszlopos listának is hívnak. Az eszköz nem javasolt státusza azt jelenti, hogy ugyan része a jelenlegi csomagnak, de a jövőben már nem fejlesztik tovább és előfordulhat, hogy az ezután megjelenő ***GTK+*** csomagokból már hiányozni fog.

A ***CList*** eszköz helyettesítésére jelenleg a ***Tree and List*** eszköz használható. A ***Ximba Radio*** működése közben gyakran van szükség a listák elemeinek dinamikus bővítésére vagy szűkítésére. Ugyanakkor sem a ***CList***, sem a ***Tree and List*** eszköz nem volt igazán alkalmas ennek az egyszerű megvalósítására. Mégis mivel a ***CList*** tervezésekor a cél kifejezetten a listák és nem a kiterjeszthető fák kezelése volt, a kettő közül ezt találtam a kevésbé bonyolultnak. Az általános meghatározás szerint egy prototípusnak az a dolga, hogy általa az adott alkalmazásból gyorsan tudjunk előállítani egy olyan működőképes változatot, amely valamilyen szokványos felülettel és funkciókkal rendelkezik. A ***Glade CList***-támogatása ezt lehetővé teszi anélkül, hogy a ***Tree and List*** eszköz összetettségével bajlódni kellene. A választás igazi haszna persze majd akkor fog megmutatkozni, amikor a ***CList*** végső elrendezésével kell foglalkoznunk.

A ***Ximba Radio*** a listákat igen intenzíven használja. Minden csatornára, kategóriára és kedvencre vonatkozó információ oszloposan elrendezett listákban jelenik meg. Míg a teljes csatornalista és a kedvencek listája statikusak, melyek soha nem változnak meg teljes egészükben, a kategórialista dinamikus. A felhasználók a kijelzőn keresztül kapcsolhatják ki vagy be ezeket, megkönnyítve ezzel a saját beállításaiknak megfelelő állomások kikeresését. A kategórialista dinamikus mivoltának kezeléséhez jó hasznát vettem a ***GLib*** két irányban láncolt listájának, a ***GList***-nek. Bármennyire összetett is egy program, csak kevés esélye van rá, hogy nélkülözze a láncolt listák megjelenését, a ***GList*** használata szerencsére meglehetősen egyszerű. Létezik az egy irányban láncolt változat is, a ***GSList***, de nem sokkal bonyolultabb a kétirányú ***GList*** használata sem. Annak a lehetősége pedig, hogy a listában mindkét irányban szabadon mozoghassunk, megér annyi fáradtságot, amennyit a ***GList*** igényelhet.

Egy másik kelepcére a program tesztelésénél számíthatunk, amennyiben ***pixmap***-eket használunk. A ***Ximba Radio*** kü-

© Kiskapu Kft. Minden jog fenntartva

4. lista Ez a függvény változtatja meg az állapotikont a kapcsolat állapotának megfelelően, a GTK+ által Apply ikonnak nevezett ikon segítségével

```
code:
gtk_image_set_from_stock(GTK_IMAGE
                         ➥(XR_Status_Image),
   GTK_STOCK_APPLY,
   GTK_ICON_SIZE_BUTTON);
```

5. lista Ez a két függvény fájlba írja ki a beállítások adatait

```
code:
void
XRUSavePrefs()
{
   ...

   /* A beállítások kiírása */
   fprintf(fd, “hostname:%s\n”, prefs.hostname);
   if ( prefs.daemondir )
      fprintf(fd,
“daemondir:%s\n”,prefs.daemondir);
   else
      fprintf(fd, “daemondir:\n”);
   fprintf(fd, “favorites:%d\n”,
      (int)prefs.enable_favorites);
   fprintf(fd, “channels:%d\n”,
      (int)prefs.channel_windows);
   fprintf(fd, “performance:%d\n”,
      prefs.performance);

   /* A kategórialista futtatása és mentése
    * az állapotukkal együtt
    */

   g_list_foreach(prefs.categories,
      SavePrefsCategory, fd);

   ...
}

static void
SavePrefsCategory(
   CatEntryT    *catentry,
   FILE         *fd
)
{
   fprintf(fd, “category:%s:%d\n”, catentry-
>name,
      catentry->state);
}
```

lönböző helyeken tüntet fel emblémaként egy-egy ***pixmap***-et. Az alapértelmezett src könyvtárból nem találja meg a program a ***pixmap***-et anélkül, hogy előtte létre ne hoznánk egy teljes fordítást:

```
./configure –prefix=<telepítési könyvtár>
make
make install
```

A folyamat során a ***pixmap*** képek átmásolódnak az előtagként megadott telepítési könyvtár alatt lévő ***pixmap*** könyvtárba. Például, ha a telepítési könyvtár a ***/usr/local/ximbaradio***, a pixmap fájlok a ***/usr/local/ximbaradio/share/ximbaradio/pixmaps*** könyvtárba kerülnek. A fentebbi telepítést elvégezve, a lefordított program rendben megtalálja a ***pixmap***-eket. Amennyiben megváltoztatjuk a ***pixmap*** fájlokat, a `make install` lépést újra meg kell tennünk. Lefordított emblémákra válthatunk – azaz a bináris állomány részévé tehetjük azokat, azért, hogy a ***pixmap***-ek helyének változása ne lehessen befolyásoló tényező. Ez elérhető a ***support.c*** fájl módosításával. A korábbi programoknál ezt az eljárást követtem, de ezen technika ismertetése már meghaladná a cikk kereteit.

Végső elemzés

A teljes programot összesen körülbelül harminc órányi munkával sikerült létrehoznom. Ennek nagy részét a központi kód írásával töltöttem. A felhasználói felület kódja nagyjából tíz óra alatt készült el. A program minden fontosabb vonatkozásában megfelel a Windowsos változat felhasználói felületének, a beállítások is könnyen kezelhetőek. A felhasználói felület kódja független a program törzsének kódjától, bár a ***callbacks.c*** fájlon keresztül még jelen van némi függőség.

A ***Ximba Radio*** fejlesztése tovább fog folytatódni, a tervek közt szerepel hangszabályozási lehetőségek és egy ***GStreamer*** alapú reflektor hozzáadása. A ***reflector***, a ***Ximba Radio*** és az ***OpenXM*** egyesítése lehetővé fogja tenni a PC-alapú ***XM Radio*** műholdas szolgáltatás távoli elérését.

A ***Glade 3*** fejlesztés alatt áll, és számíthatunk arra, hogy a kódgeneráló részt majd eltávolítják. Mivel már elég régóta fejlesztik ezt a változatot, és a kibocsátása sem tűnik túl közelinek, az általunk használt program által létrehozott kód felhasználása a ***Glade***-del való prototípuskészítéshez a közeljövőben még biztosan életképes lehetőség marad. Vagyis a prototípusfejlesztés továbbra is egyszerű és gyors a ***GTK+*** és a ***Glade*** felhasználásával.

Linux Journal 2004. szeptember, 125. szám

Michael J. Hammel szoftvermérnök és író, a texasi Houstonban él Brinda nevű feleségével és Ryann nevű lányával. A fáradhatatlan futó és teniszjátékos, aki odáig van a kutyáiért, Reba-ért és Baley-ért, valamint a számítógépéért. Szabadidejét öregedő karfák ápolásával és szakadt kanapépárnák tisztításával tölti, és azt kérdezi magától, hogy miért nincs egy perc szabadideje sem.
Honlapja a ➲ graphics-muse.com címen érhető el, ő maga pedig a mjhammel@graphics-muse.org levélcímen.