Grafikus felület programozása QT-ban (2. rész)

Folytassuk grafikus felülettel rendelkező programunk fejlesztését! Kibővítjük szolgáltatásainak a körét, valamint több nyelv támogatására készítjük fel.

Cikkünk 2002 októberében megjelent első részében a Qt Designerrel létrehoztunk egy aprócska szövegszerkesztő programot. A most következő oldalakon a program működésbeli hiányosságait fogjuk pótolni, majd a `make` és a g++, C++ fejlesztőeszközök segítségével felkészítjük a többnyelvű használatra. A fordítási munkához vonzó munkakörnyezetként a Qt Linguistet fogjuk használni. A Qt-alkalmazásunkhoz szükséges `Makefile` elkészítéséhez a `qmake` segédprogramot használjuk, ami egyre inkább kiszorítja a lassan elavulttá váló `tmake`-et. Az említett `tmake` a Qt régebbi változataiban használatos, Perlben írt eszköz volt.
Nézzük, hol tartunk! Noha szövegszerkesztőnk grafikus felülete már képes a megnyitott ablakok bezárására és az alkalmazásból való kilépésre, még mindig nélkülöznünk kell például a párbeszédablakot, amelyben a felhasználó dönthet az adatok mentése vagy a korábban mentett állomány megtartása mellett. Az ***Új***, ***Mentés*** és ***Mentés másként*** menüpontok egyelőre még nem érhetők el, a program viszont támogatja a másolási, kivágási és beillesztési műveleteket, valamint a műveletek visszavonását és megismétlését. Ezenkívül képes dőlt, aláhúzott és félkövér betűjellemzőket is használni, sőt ezek tetszőleges kombinációira is.
A ***Súgó*** menüben a ***Névjegy*** csak akkor válik teljes mértékben működőképessé, ha a grafikus felületet befordítjuk a C++-programba. A még hiányzó szolgáltatásokat a Qt Designer saját szerkesztőjével is elkészíthetjük. Minthogy a Qt Designerben nem lehet egyszerűen a teljes projektet lefordítani, talán könnyebb, ha mégis a kedvenc szerkesztőnket használjuk.

Szerkesztőnk összeépítése

Mivel mostanra elkészültünk a felhasználói felület leírását tartalmazó ***.ui*** állománnyal és a megírt forráskódot tartalmazó ***ui.h*** állománnyal, nézzük, mi hiányzik még!
Legelőször az `uic` eszközzel (user interface compiler) az XML-ből C++-kódot kell készítenünk, de szerencsére a `qmake` leveszi vállunkról ezt a terhet. Egyszerűen hívjuk meg a `qmake` fordítót a Designer által létrehozott projektfájllal:

```
qmake -o Makefile lj-article.pro
```

A `Makefile` az ***.ui*** nevű alkönyvtárral együtt készül el, amely az ***ljeditor.h*** és az ***ljeditor.ui.h*** állományokból létrehozott C++-forráskódok tárolására hivatott.
Ha ez nem történik meg, ellenőrizzük a `QMAKEPATH` változó beállítását: a használandó Qt-környezet ***mkspecs*** alkönyvtárára kell mutatnia:

```
QMAKEPATH=$QTDIR/mkspecs/linux-g++
export QMAKEPATH
```

Nézzük meg azt is, hogy a ***$QTDIR/bin*** könyvtár szerepel-e a keresési útvonalban. A C++-állományok létrehozása az alábbi módon történik:

```
make .ui/ljeditor.h
make .ui/ljeditor.cpp
```

Ha hiba jelentkezne, ellenőrizzük a `QTDIR`, `PATH` és `LD_LIBRARY_PATH` változók értékeit. Az elsőnek a Qt-környezet ***lib*** és ***include*** könyvtárainak szülőkönyvtárára szükséges mutatnia. A `uic`, `qmake` és Designer programok könyvtárának szerepelnie kell a `PATH` változóban, míg a ***$QTDIR/lib*** szükséges az összeépítő (linker) számára.
Arra figyeljünk, hogy a két új fájlba közvetlenül felvitt változások elvesznek, ha az ui-fájlt a Designerben szerkesztjük, és újabb fordítási kör is szükséges lesz. Ezért inkább származtassunk egy új osztályt az ljeditből, és abba készítsük el a kívánt változásokat. Az uic két kapcsolója, a `-subdecl` *`osztaly_nev`* és a `-subimpl` *`osztaly_nev`* segítségével létrehozhatjuk az új forráskódvázat. Az alábbi paranccsal

```
uic -o editor.h -subdecl Editor
➥.ui/ljeditor.h ljeditor.ui
```

hozzájutunk az ***editor.h*** állományhoz, ami tartalmazza az új Editor-gyermekosztályhoz szükséges függvény-prototípusokat. A következő parancs pedig létrehozza az ***editor.cpp*** állományt a szükséges megvalósítás vázával (figyeljük meg, hogy kapcsolóként a fejállományt is megadtam):

```
uic -o editor.cpp -subimpl Editor
➥editor.h ljeditor.ui
```

Ezt a két új állományt hozzá kell adnunk a projekthez, vagyis az ***lj-article.pro*** állományhoz, az alábbi két sor segítségével:

```
HEADERS += editor.h
SOURCES += editor.cpp
```

Ugyanezt a Qt Designerben is megtehetjük, ha megnyitjuk a projektfájlt, majd a ***Project***, ***Add File*** menüpontot használva kiválasztjuk a két fájlt. Figyeljünk rá, hogy beillesztéskor az állományok típusát (C++) is megadjuk!
Az uic-eszköz által létrehozott kód tartalmazni fogja a szülőosztályban levő összes függvény vázát. Ezért valóban helyénvaló, ha töröljük azokat a függvénymeghatározásokat és -vázakat, amelyeket a részosztályban már nem kívánunk újra létrehozni. Az ***editor.h*** állományból távolítsuk el ezeket a sorokat:

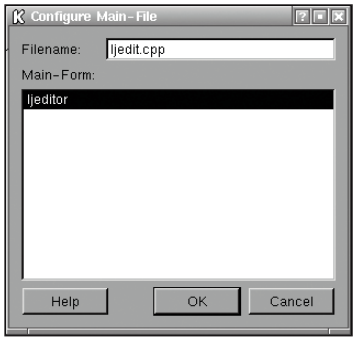
```
void fileExit();
void helpAbout();
void fileClose();
```

Ezután az ***editor.cpp*** állományból is töröljük a hozzájuk tartozó kódvázakat, amelyek valahogy így néznek ki:

```
void Editor::fileExit()
```

```
{
    qWarning( "Editor::fileExit() "
              "not yet implemented!" );
}
```

Természetesen a teljes programhoz még szükségünk van a `main()` függvényre. Ez megírható akár közvetlenül forráskódban, de a Qt Designer tud egy kicsit segíteni. Válasszuk a ***File, New, C++ Main file*** menüpontot, majd töltsük ki a megjelenő ablakot. A következő párbeszédablakban (lásd *1. kép*) adhatunk nevet a főprogramnak (én az ***ljedit.cpp*** nevet választottam) és a főelemnek (main widget). A Designer itt nem kínálja fel az Editor-alosztályt, így nem sok választásunk van, döntsünk az ljeditor mellett. Ennek a névnek a kiválasztása azt jelenti, hogy a létrehozott kódot módosítani kell. Az ***ljeditor.h*** helyett az ***editor.h***-t kell használjuk, és amikor új objektumot kell létrehozni, nem az ljeditorból, hanem az Editorból kell példányosítanunk. A változtatások után ***ljedit.cpp*** programunk a következőképpen fog kinézni:



1. kép A main( ) főfüggvény

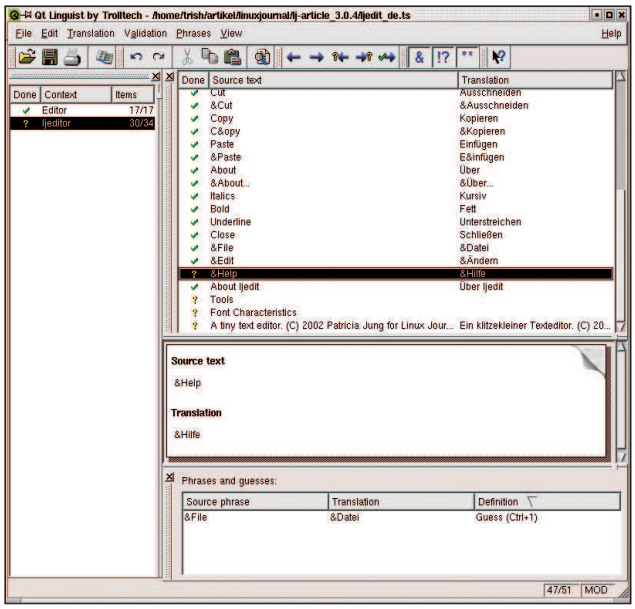
```
#include <qapplication.h>
#include "editor.h"

int main( int argc, char ** argv )
{
    QApplication a( argc, argv );
    Editor *w = new Editor;
    w->show();
    a.connect( &a, SIGNAL( lastWindowClosed() ),
               &a, SLOT( quit() ) );
    return a.exec();
}
```

Amint az minden Qt `main()` függvényben megszokott, létrehozunk egy `QApplication` objektumot, átadjuk neki parancssori értékeket (argv), majd létrehozzuk a *w* nevű Editor-elemet. Ezt követően megmutatjuk a világnak, és az `a.exec()` függvénnyel belépünk a tényleges programba.
Talán magad is észrevetted, hogy az alkalmazás főelemét kijelölő `a.setMainWidget ( w );` sor hiányzik – erre később még kitérek. Mivel főelem nélkül az alkalmazás nem képes kilépni, az alkalmazás `lastWindowClosed()` jelét annak `quit()` foglalatába kell kötni.
A `make`, majd a projektkönyvtárban kiadott `./lj-article` indítóparancs hatására már egy futásra kész, de még nem teljes értékű szövegszerkesztőt láthatunk.
Ha a projektállomány alapelnevezésétől eltérő nevet szeretnénk választani, az ***lj-article.pro*** projektállományt egészítsük ki a `TARGET=ljedit` sorral.

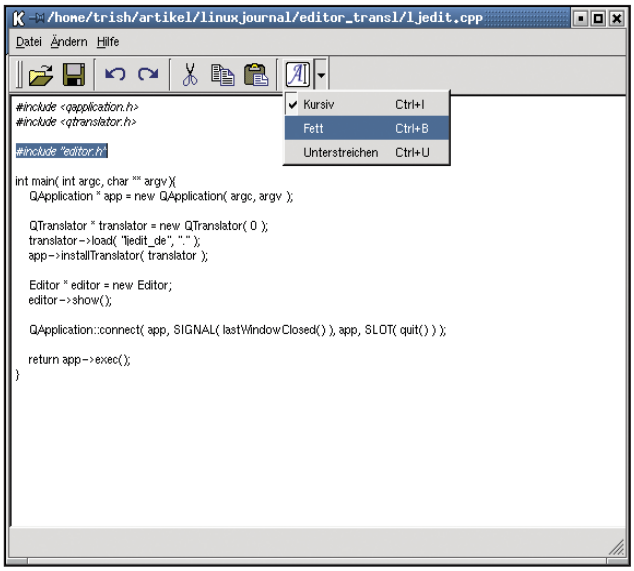
Jöjjön a kódolás!

Alapvetően nem teszünk sokkal többet, minthogy az ***editor.cpp***-ben a `qWarning "Not yet implemented"` kiírását valamilyen hasznosabb tevékenységgel helyettesítjük.



2. kép A fordítók barátja: a Qt Linguist

Amennyiben a Designer szerkesztője megfelel a céljaidnak, a különböző alosztályok miatt nem különösebben kell aggódnunk, de a fordítás és hibakeresés fárasztóvá válhat, ezért én nem ezt a megoldást javaslom.
Az 1. lista az ***editor.h*** tartalmát mutatja be, a második lista pedig kivonat az ***editor.cpp*** programból. (Valamennyi lista letölthető a ➲ http://www.linuxvilag.hu/qt oldalról, 4722.tar.gz néven.)
A maradék négy foglalat mellett (amelyeket még nem készítettünk el) újraírjuk a `closeEvent()` függvényt, itt kérdezünk rá a felhasználótól, hogy tényleg be kívánja-e zárni az ablakot.
Az egyértelműség érdekében az idevágó párbeszédablak önálló függvényben, a `saveAndContinue()`-ban kapott helyet.
Az eddig említett változókon kívül két további változóra is szükségünk lesz: a `fileName`-re, ami az éppen szerkesztés alatt álló állomány nevét tartalmazza, és az `editFieldre`, a `QTextEdit`-elem másolatának tárolására. A szerkesztő részét képező (az objektumközpontú programozásnak megfelelő) létrehozó (constructor) egyetlen feladata, hogy ezeknek a változóknak kezdőértéket adjon.
Egy másik könnyű feladat a `fileNew()` foglalat megvalósítása. Ez a függvény új szerkesztőablakot hoz létre, majd meg is jeleníti. Ez az oka annak, hogy az első szerkesztőablakot nem tesszük meg mindjárt az alkalmazás főelemévé, hiszen ennek bezárása az alkalmazás összes többi ablakának bezárását is magával vonná.
De vajon mi játszódik le akkor, amikor a felhasználó bezárja az egyik ablakot vagy éppen magát az alkalmazást? A kibővített `closeEvent()` az átadott értékekkel együtt meghívja a `saveAndContinue()` függvényt: voltaképpen itt kérdezünk rá a felhasználótól arra, hogy tényleg ki akar-e lépni (159. sor).
Amint az üzeneteknél szokás, itt is `tr()` függvényen keresztül adjuk át a szöveget, így segítjük a program más nyelvekre történő fordítását.
Ha a `saveAndContinue()` függvény beleegyező választ ad vissza, a folyamat folytatódik, egyébként az eseményt eldobjuk.
Ha a szövegszerkesztő tartalmának a felhasználó már állománynevet is adott, vagy legalábbis már gépelt szöveget a `QTextEdit` elembe, helyes azt feltételezni, hogy a munkát meg szeretné tartani. A `saveAndContinue()` ekkor az állo-



3. kép Az ljedit.cpp (nyelvi lehetőségek kihasználása – német)

mánynevet az ablak címsorában feltüntetve üzenetablakot jelenít meg, és a mentés felől érdeklődik. Itt három gomb közül kell választani: ***Yes***, ***No*** és ***Cancel***.
A `fileName` kiírását a `%1` helyőrző segítségével oldjuk meg, ez azért fontos, mert más nyelveknél a szórend változása miatt esetleg az üzeneten belül máshova kerül a fájlnév. Ha a felhasználó a ***Yes*** feliratú gombot nyomta meg (181. sor), akkor a szerkesztő tartalmának mentése a megadott név alatt történik meg. Abban az esetben, ha eddig még nem kaptunk állománynevet, a `fileSaveAs()` függvény a mentés megkezdése előtt rákérdez a kívánt értékre. Ha viszont a válasz a ***No*** gomb megnyomása volt, akkor nem mentünk. A felhasználó minderről 2000 ezredmásodpercenként az állapotsorban fog értesítést kapni (187. sor).
A ***Cancel*** gomb hatására egy üzenet jelenik meg az állapotsorban, a `saveAndContinue()` függvény pedig a „nem, ne folytasd” üzenetet adja vissza. Ha az állománynév sincs megadva és a szerkesztői ablak is üres, az üzenetablak meg sem jelenik. A visszatérési érték ebben az esetben ***Igaz*** (TRUE) lesz: „mindenképp folytasd” (201. sor).
A `saveAndContinue()` függvény hívása a `fileOpen()` foglalatból is történhet. Ha szerkesztőablak már tartalmaz szöveget, vagy az állománynév meghatározása már korábban megtörtént, a felhasználónak lehetősége van e tartalmat lemezre menteni, mielőtt ugyanebben az ablakban egy másik állományt megnyitna.

A világraszóló program

A szöveges tartalmak `tr()`-be ágyazásával lehetőségünk nyílik a programot más nyelvek használatára is felkészíteni. Ez a lépés a `main()` függvény csekély módosítását és a tényleges fordítási munka elvégzését igényli. Ez utóbbi könnyedén elvégezhető a Qt Linguist használatával (*2. kép*).
Mielőtt azonban saját magad nekilátnál vagy más szakfordító(k) nekifognának a program más nyelvekre való átültetéséhez, először is magukat a lefordítandó szövegrészeket kell beszereznünk. Emiatt újabb változóval – a `TRANSLATIONS`-szel – kell a ****.pro*** állományt bővíteni:

```
TRANSLATIONS = ljedit_de.ts ljedit_no.ts
```

Az iménti példa azt mutatja be, hogy ennek az alkalmazásnak a német, illetve a norvég fordítása fog megtörténni. Fontos megjegyezni, hogy az adott nyelvre jellemző fordításokat az állománynév végén szereplő országrövidítésekkel („_de”, illetve „_no”) jelöljük. Amint kijelöltük a kívánt nyelveket az alkalmazásunk számára, az

```
lupdate lj-article.pro
```

parancs a nyelvészprogram számára létrehozza az XML-állományokat. A fordító most egymás után lefordítja a megadott karakterláncokat, és a munka előrehaladtával a sárga kérdőjelek zöld pipákra változnak.
Ha a bal oldalon lévő panelen látható összes osztály mellett megjelenik a zöld pipa – a fordítással készen vagyunk.
A hivatásos fordítók először egy kifejezéspárokból álló szótárat fordítanak le, mielőtt a program tényleges fordítását elkezdenék. A Linguist jelenleg nem támogatja a kifejezésszótárak használatát, sem teljes kifejezések fordításának bemásolását (például egy külső fordítóprogramból), így egy nagyobb program fordítási feladata rémisztően hathat a gyakorlatlan fordítókra.
A programban viszont hasznos minőségellenőrzéssel kapcsolatos szolgáltatásokat találunk: a ***Validation***, ***Accelerators*** menüpont (ha engedélyezzük) biztosítja, hogy az „&” jellel megadott gyorsbillentyűk a fordításokban is szerepelnek.
A ***Validation***, ***Ending punctuation*** pedig biztosítja, hogy a fordítás azonos írásjellel végződjön, mint az eredeti szöveg.
Amint a legutolsó lefordított kifejezés mentése is megtörtént, a ***File***, ***Release…*** menü a fordításokat ****.qm***-kiterjesztésű, a program által használható bináris állománnyá alakítja. Ha több fordítást tartalmazó ****.ts*** fájlt szeretnénk terjeszteni, a

```
lrelease lj-article.pro
```

parancs segít. Amennyiben programmódosítás még ezt követően is történik, az lupdate program a projektbe felvett ****.ts*** állományokba átviszi a változtatásokat, az lrelease pedig frissíti a bináris változatot.
A 3. lista a már nyelvi környezetet is megfelelően használni tudó `main()` függvényt mutatja be. A program újdonsága a két beillesztett állomány: a ***qtranslator.h*** és a ***qtextcodec.h***.
A használt nyelv a rendszer nyelvi beállításától függően (a `LANG` környezeti változóban szerepel) kerül kiválasztásra (13. sor). Ha például a `LANG` változónak a `de` vagy `de_DE` értéket adjuk, az alkalmazás a ***/local/lib*** könyvtárban az ***ljedit_de*** vagy ***ljedit_de.qm*** nevű állományokat fog keresni. Amennyiben ilyeneket mégsem találna, a program az eredeti nyelvi változatot fogja használni. Sajnos nincs egyszerű megoldás a több könyvtárra kiterjedő keresésre, sem pedig a programba épített könyvtárnevek módosítására.
Ha a QTranslator megtalál egy fordításállományt, akkor betölti azt, és onnantól kezdve a fordításokat használja (15. sor).
Az ljedit német változatát a *3. képen* láthatjuk.

Linux Journal 2002. október, 102. szám

Patricia Jung
(trish@trish.de) rendszergazda, szakíró és szerkesztő egy személyben, és ilyen minőségében örömmel tölti el, hogy élhet előjogaival és kizárólag Unix-, illetve Linux-rendszerekkel foglalkozhat.