KDE programming tutorial using KDevelop

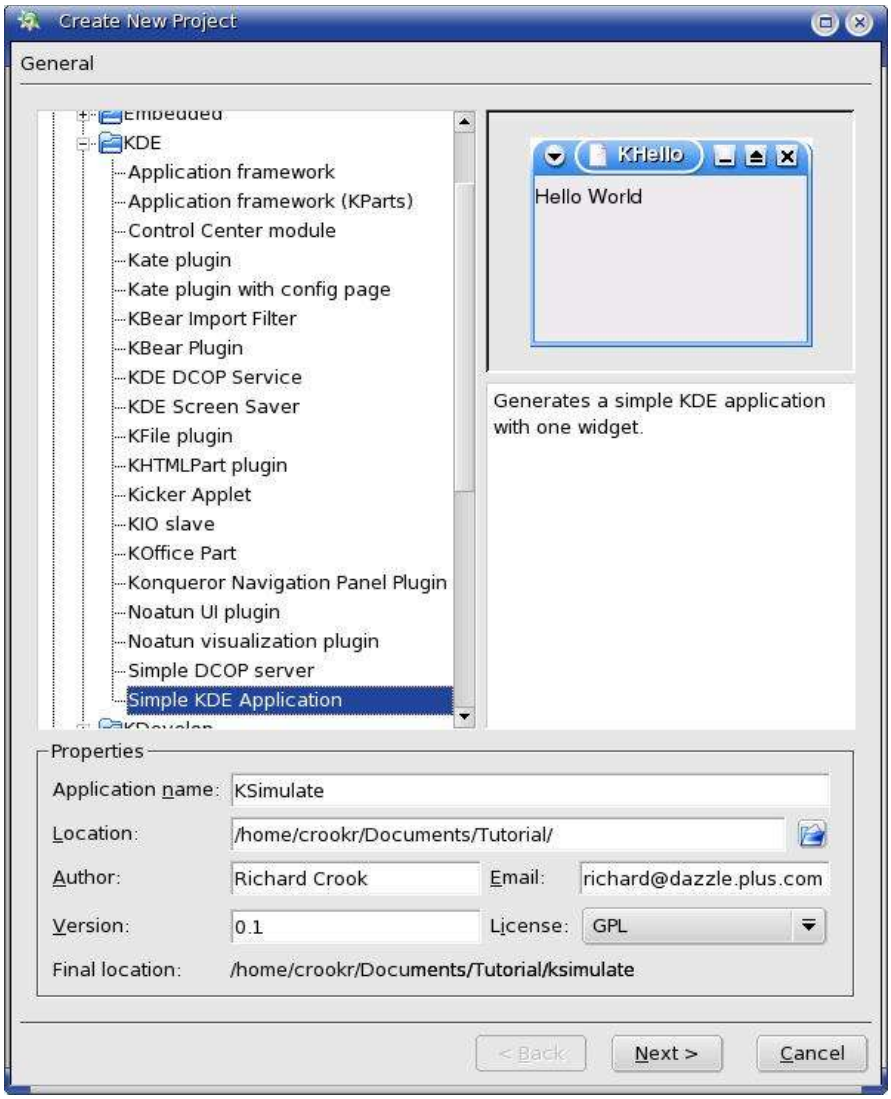
Tutoriál od Richarda Crooka se snažil přeložit a trochu okomentoval Juraj Václavík

Poznámka překladatele (dále jen PP): Tento překlad vznikl za účelem vlastního pochopení práce se systémem KDevelop, přičemž samotný autor se sice trochu programováním zabýval, ale „neobjektivním“ a neznajíce C - asi bude nemálo významových chyb.

Vzorová aplikace bude simulovat síť paketového rozhlasu s uživatelským rozhraním, ale berte to prosím jako demonstraci programátorských technik, nikoliv jako smysluplnou aplikaci.

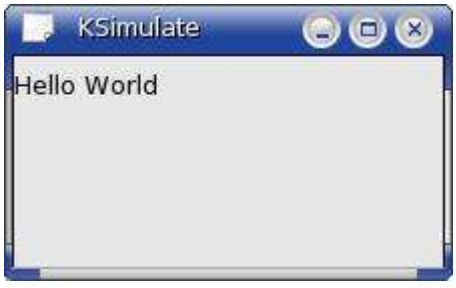
1. Nový projekt

Po spuštění KDevelopu z nabídky Project vybereme „New Project“ a otevřeme stromovou strukturu. Z ní vybereme v C++/KDE "Simple KDE Application", zadáme jméno aplikace, kde ji chceme uložit a přest Next dojdeme k Finish.
KDevelop se nyní nastaví pro vývoj KDE aplikací a otevře nějakou šablonu.



Překlad a spuštění kódu

Než se zahájí překlad je nutno spustit automake a autoconf. Nejprve spustíme automake volbou "Run automake & friends" z Build menu. Potom spustíme autoconf volbou "Run Configure" z Build menu.
PP: pokud to neuděláme a zkusíme rovnou překládat, nabídne nám tyto úkony KDevelop sám.
Nyní můžeme překládat volbou "Build Project" a spustit aplikaci volbou "Execute Program", buď z Build menu, nebo ikonou v nástrojové liště anebo klávesovou zkratkou (PP: viz „Configure shortcuts“ v Settings menu).



2. Šablona - změna kódu

Tento příklad vychází ze šablony, v které postupně budeme nahrazovat výchozí kód. Soubory otevřeme v menu File "Quick Open..." a nahradíme výchozí kód následujícím:

main.cpp

```
#include "ksimulate.h"
#include <kapplication.h>
#include <kcmdlineargs.h>
#include <kaboutdata.h>

// structure that holds command line options
static KCmdLineOptions options[] = { KCmdLineLastOption };

int main(int argc, char **argv)
{
  KAboutData about("ksimulate",                  // appName - program name used internally
                   "KSimulate",                  // programName - displayable progame name
                   "0.1",                        // version - program version string
                   "Simulates radio networks",   // shortDescription - what program does
                   KAboutData::License_GPL,      // licenceType
                   "(C) 2004 Richard Crook",     // copyrightStatement
                   "Example KDE application!",   // text - any information
                   "http://www.dazzle.plus.com", // homePageAddress
                   "richard@dazzle.plus.com");   // bugsEmailAddress

  about.addAuthor( "Richard Crook", "Author", "richard@dazzle.plus.com" );

  // uses KAboutData to replace some of the arguments that would otherwise be required
  KCmdLineArgs::init(argc, argv, &about);
  KCmdLineArgs::addCmdLineOptions( options );

  // controls and provides information to all KDE applications
  KApplication app;
  KCmdLineArgs *args = KCmdLineArgs::parsedArgs();

  // create main window, enter main event loop, and wait until exit() is called
  KSimulate *mainWin = new KSimulate();
  app.setMainWidget( mainWin );
  mainWin->show();
  args->clear();
  return app.exec();
}
```

ksimulate.h

```
#ifndef KSIMULATE_H
#define KSIMULATE_H

#include <kmainwindow.h>

/**********************************************************/
/********** KSimulate is the main application window*******/
/**********************************************************/

class KSimulate : public KMainWindow
{
  Q_OBJECT
public:
  KSimulate();                              // constructor
};

#endif // KSIMULATE_H
```

ksimulate.cpp

```
#include "ksimulate.h"

/**********************************************************/
```

```
/********** KSimulate is the main application window ************/
/*****************************************************************/

/************************ constuctor ***************************/

KSimulate::KSimulate() : KMainWindow()
{
}
#include "ksimulate.moc"
```

Nakonec za pomoci Automake Manager odstraníme "ksimulateui.rc", protože ho nebudeme potřebovat. Automake Manager se otevírá pomocí ikony zcela vpravo. Doporučuji také odstranit z disku, protože tento soubor nebude zapotřebí. PP: Nachází se v src -> data in shellrc.

Průzkum kódu

Aplikace se skládá z "main" rutiny v main.cpp (vstupní bod, odkud je aplikace spouštěna) a třídy "KSimulate" v ksimulate.h a ksimulate.cpp (hlavní okno aplikace).
Doufám, že komentáře v kódu napoví, co rozličné části kódu dělají, but lets go through the code in a bit more detail.

main.cpp

```
#include "ksimulate.h"
#include <kapplication.h>
#include <kcmdlineargs.h>
#include <kaboutdata.h>
```

Klauzule Include: ksimulate.h poskytuje přístup ke třídě hlavního okna, kapplication.h zajišťuje základní funkcionalitu požadovanou všemi KDE aplikacemi, kcmdlineargs.h je třída pro příkazový řádek a argumenty a kaboutdata.h je třídou pro zápis informací o aplikaci.

```
// structure that holds command line options
static KCmdLineOptions options[] = { KCmdLineLastOption };
```

Zde můžeme přidat další specifické argumenty pro příkazový řádek, zde ponecháme výchozí.

```
  KAboutData about("ksimulate",                  // appName - program name used internally
                   "KSimulate",                  // programName - displayable progame name
                   "0.1",                        // version - program version string
                   "Simulates radio networks",   // shortDescription - what program does
                   KAboutData::License_GPL,      // licenceType
                   "(C) 2004 Richard Crook",     // copyrightStatement
                   "Example KDE application!",   // text - any information
                   "http://www.dazzle.plus.com", // homePageAddress
                   "richard@dazzle.plus.com");   // bugsEmailAddress

  about.addAuthor( "Richard Crook", "Author", "richard@dazzle.plus.com" );
```

Toto je záznam informací o aplikaci.

```
  // uses KAboutData to replace some of the arguments that would otherwise be required
  KCmdLineArgs::init(argc, argv, &about);
  KCmdLineArgs::addCmdLineOptions( options );
```

Zde inicializujeme třídu argumentů příkazového řádku a přidáváme výchozí nastavení.

```
  // controls and provides information to all KDE applications
  KApplication app;
  KCmdLineArgs *args = KCmdLineArgs::parsedArgs();
```

Zde vytváříme objekt KApplication vybavený základní funkcionalitou požadovanou všemi KDE aplikacemi a parsující příkazový řádek.

```
  KSimulate *mainWin = new KSimulate();
  app.setMainWidget( mainWin );
```

Nyní vytvoříme hlavní okno aplikace a připojíme ho k objektu KApplication.

```
  mainWin->show();
```

Zviditelníme hlavní okno.

```
  args->clear();
  return app.exec();
```

Vymažeme argumenty, aby se uolnila paměť a and pass control to the KApplication object until our application is closed.???

ksimulate.h

```
#ifndef KSIMULATE_H
#define KSIMULATE_H
```

Zpracuj obsah pouze, pokud není symbol definován.

```
#include <kmainwindow.h>
```

Soubor kmainwindow.h file poskytuje třídu KMainWindow která bude základem třídy okna aplikace.

```
class KSimulate : public KMainWindow
{
  Q_OBJECT
public:
  KSimulate();                    // constructor
};
```

Aby zdědila všechny potřebné KDE vlastnosti je naše třída je odvozena od KMainWindow. Q_OBJECT je zvláštní makro potřebné pro vyvolání meta object compileru (ten je součástí Qt development library), které se musí zahrnout do všech KDE tříd. Nyní obsahuje pouze metodu konstruktoru.

```
#endif // KSIMULATE_H
```

Konec bloku.

ksimulate.cpp

```
#include "ksimulate.h"
```

Zahrň hlavičkový soubor.

```
KSimulate::KSimulate() : KMainWindow()
{
}
```

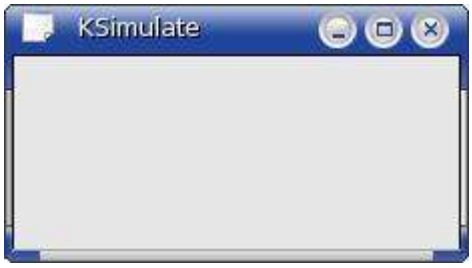
Implementuj (zatím prázdnou) třídu konstruktoru zděděnou z KMainWindow.

```
#include "ksimulate.moc"
```

Meta object compiler vytvoří potřebný .moc soubor. Moc soubor obsahuje meta objekt kód pro třídy používané makrem Q_OBJECT. Mezijiným meta object code jsou potřebné pro mechanizmus signálů a slot (použijeme později).

Překlad a spuštění kódu

Nový kód se překompiluje sám, když znovu spustíme aplikaci v KDevelopu.
Také zkuste spustit aplikaci z příkazové řádky zadáním „./ksimulate --help“. Aplikaci najdete v adresáři "ksimulate/debug/src".



3. Přidání (řádku) nabídky a stavového řádku

Třída hlavního okna "KSimulate" je odvozena od KMainWindow a může dědit mnoho užitečných zařízení (vlastností) usnadňující vybudování aplikace (dost volně).

ksimulate.cpp

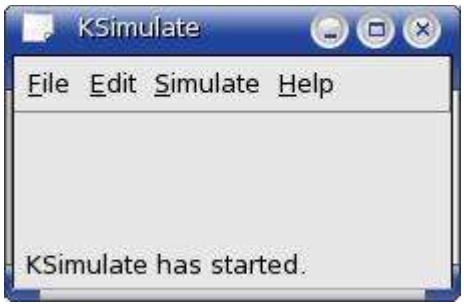
Nejprve potřebujeme zahrnout hlavičky 3 tříd: kstatusbar.h pro přístup k funkcionalitě stavového řádku, kmenubar.h pro hlavní menu na horní straně hlavního okna, a kpopupmenu.h pro vytvoření otevírajícího se menu.

```
#include <kstatusbar.h>
#include <kpopupmenu.h>
#include <kmenubar.h>
```

Potom do KSimulate konstruktoru přidáme kód pro menu a stavový pruh.

```
// create drop-down menus
KPopupMenu *menuFile = new KPopupMenu( this );
KPopupMenu *menuEdit = new KPopupMenu( this );
KPopupMenu *menuSim  = new KPopupMenu( this );

// add drop-down menus to main menu bar
menuBar()->insertItem( "&File", menuFile );
menuBar()->insertItem( "&Edit", menuEdit );
menuBar()->insertItem( "&Simulate", menuSim );
menuBar()->insertItem( "&Help", helpMenu() );

statusBar()->message("KSimulate has started.");
```

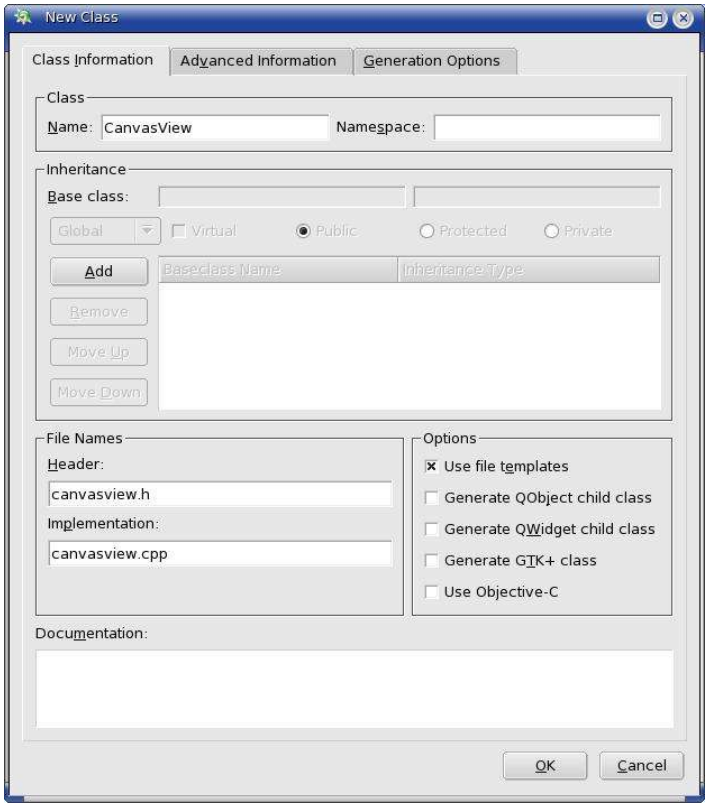
Vytvoří se 3 otevírající se menu (File, Edit a Simulate) nyní bez položek - později přidáme. Menu se budou otevírat. Otevírající se menu jsou připojena k hlavnímu menu pomocí KMainWindow a je připojena nabídka speciální help. Nakonec zobrazíme zprávu ve stavovém řádku. Stavový pruh a menu je vytvořeno samočinně KMainWindow při spuštění.



Překlad a spuštění kódu

Nový kód po spuštění aplikace... Vyzkoušejte chování menu.

4. Co je canvas



Canvas je optimalizovaná 2D grafická plocha, která může obsahovat libovolné množství grafických prvků. Grafické prvky mohou mít libovolný tvar, velikost a obsah a mohou se po canvas volně pohybovat. Canvas použijeme ke konstrukci grafické ploch aplikace.
Modul canvas používá model dokumentu/ pohledu. Třída pohledu canvas se používá k zobrazení částečného pohledu na canvas. V jednom okamžiku může nad jedním canvas operovat vícero pohledů (?). Použijeme pohled na canvas k zobrazení naší hlavní grafické oblasti v hlavním okně aplikace. The canvas module uses a document/view model. A canvas view class is used to show a particular view of a canvas. Multiple views can operate on the same canvas at the same time.

Přidání nové třídy

Náš pohled na plátno bude realizován jako potomek třídy QcanvasView, aby zdědila základní sadu funkcionality.
Nová třída se bude nazývat "CanvasView" a přidá se do projektu přes průvodce z nabídky Project -> "New Class...". Zadejte jméno třídy, stiskněte "OK", a v následujícím dialogu zaškrtněte "Do not ask me again and use always my Active Target" a znova "OK".

Aby se kód stal snadněji čitelný a lépe plnil naše požadavky, budeme znovu nahrazovat kód šablony kódem naším, uvedeným níže. Otevřeme každý soubor z nabídky File -> "Quick Open..." a nahradíme existující kód kódem uvedeným níže.

canvasview.h

```
#ifndef CANVASVIEW_H
#define CANVASVIEW_H

#include "ksimulate.h"
#include <qcanvas.h>
#include <qevent.h>

/*****************************************************************************/
/******************** CanvasView provides access to the canvas ***************/
/*****************************************************************************/

class CanvasView : public QCanvasView
{
  Q_OBJECT
public:
  CanvasView( QCanvas*, KSimulate* );              // constructor

protected:
  void viewportResizeEvent( QResizeEvent* );       // view resized
};
```

```
#endif // CANVASVIEW_H
```

Jak bylo dříve uvedeno, naše nová třída je odvozena z třídy QCanvasView a dědí základní funkcionalitu. My nyní budeme funkcionalitu rozšiřovat a implementovat virtuální chráněné metodu pro přijetí události pro změnu velikosti z hlavního okna aplikace.

canvasview.cpp

```
#include "canvasview.h"
/*****************************************************************************/
/****************** CanvasView provides access to the canvas *****************/
/*****************************************************************************/

/************************** constuctor ***************************************/

CanvasView::CanvasView(QCanvas* canvas, KSimulate* parent)
           : QCanvasView( canvas, parent )
{
}

/************************* viewportResizeEvent *******************************/

void CanvasView::viewportResizeEvent( QResizeEvent *event )
{
  // resize canvas
  canvas()->resize( event->size().width(), event->size().height() );
}

#include "canvasview.moc"
```

V metodě viewportResizeEvent změníme velikost tak, aby přesně zaplnila hlavní okno (viewport). Viewport je na hlavním okně plocha mezi řádkem nabídky nahoře a stavovým řádkem dole.

Vytvoření canvas a pohledu na canvas

Plátno a pohled na plátno se musí vytvořit jako část konstruktoru okna hlavní aplikace.

ksimulate.cpp

Do hlavičkového souboru:

```
#include "canvasview.h"
```

A do konstruktoru přidejte kód, který vytvoří plátno a pohled na plátno a nastaví pohled na plátno jako ústřední widget (?).

```
  // create canvas and canvas view
  QCanvas*     canvas      = new QCanvas( this );
  CanvasView* canvasView  = new CanvasView( canvas, this );
  setCentralWidget( canvasView );
  canvasView->show();
```

Překlad a spuštění kódu

Protože jsme přidali novou třídu, musíme znovu spustit automake z nabídky Build -> "Run automake & friends". Potom se spustí nový kód.

5. Rozšíření canvas

V této části chceme dosáhnout 2 věcí. Za prvé ovládat canvas, aby se nezmenšilo pod nějakou minimální velikost, a za druhé umožnit pohledu na canvas aktualizovat stavový řádek.
Minimální velikost canvas se ovládá 2 proměnnými ze třídy CanvasView class. K zajištění vhodného přístupu ke stavovému řádku bude třída CanvasView nastavovat privátní ukazatel.

Aktualizace CanvasView

canvasview.h

Do deklarace třídy CanvasView Přidáme 2 privátní proměnné k zaznamenání minimální šířky a výšky canvas, 2 konstanty obsahující výchozí minimální hodnoty šířky a výšky a ukazatel pro přístup ke stavovému řádku.

```
private:
  KStatusBar*          sbar;                  // shortcut to KSimulate's statusbar
  int                  minCanvasW;            // minimum canvas width
  int                  minCanvasH;            // minimum canvas height
  static const int     MIN_CANVAS_W = 350;    // initial canvas minimum width
  static const int     MIN_CANVAS_H = 200;    // initial canvas minimum height
```

canvasview.cpp

Včlenění kstatusbar.h pro přístup k funkcionalitě stavového řádku.

```
#include <kstatusbar.h>
```

V konstruktoru CanvasView inicializujeme privátní ukazatel a minimální šířku a výšku.

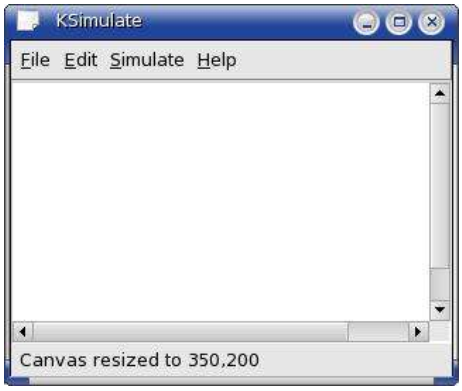
```
  // set statusbar shortcut
  sbar = parent->statusBar();

  // set initial minimum canvas size
  minCanvasW = MIN_CANVAS_W;
  minCanvasH = MIN_CANVAS_H;
```

Nahradíme kód metody viewportResizeEvent method, aby velikost canvas nebyla nikdy menší, než minimální velikost a aktualizujeme stavový řádek.

```
  int w = event->size().width();
  int h = event->size().height();
  if ( w < minCanvasW ) w = minCanvasW;
  if ( h < minCanvasH ) h = minCanvasH;

  // resize canvas
  canvas()->resize( w, h );

  // update status bar to say resize happened
  sbar->message(QString("Canvas resized to %1, %2").arg(w).arg(h));
```

Překlad a spuštění kódu

Nový kód bude samočinně přeložen při prvním spuštění aplikace. Ověřte chování aplikace.



6. Kreslení na canvas

Položky canvas

Dříve jsem uvedl, že na canvas lze zobrazit libovolné množství grafických prvků. Qt poskytuje řadu předdefinovaných prvků a další specializovanější prvky lze vytvořit tak, že vytvoříme potomky stávajících prvků. K zobrazení naší sítě stanic vytvoříme novou třídu odvozenou z QCanvasSprite.

Přidání nové třídy

Naše nová třída se bude nazývat "Station" a přidá se do projektu stejně, jako jsme dříve přidali průvodce CanvasView. "New Class...", zadáme jméno a "OK".
Abychom nad kódem snadněji udrželi kontrolu, znovu nahradíme výchozí kód KDevelopu kódem uvedeným dále. Otevřeme každý soubor z nabídky File -> "Quick Open..." a nahradíme stávající kód novým.

station.h

```
#ifndef STATION_H
#define STATION_H

#include <qcanvas.h>

/*****************************************************************************/
/************************** Station to be simulated **************************/
/*****************************************************************************/

class Station : public QCanvasSprite
{
public:
  Station( QCanvasPixmapArray*, QCanvas*, int, int );       // constructor
};

#endif // STATION_H
```

Naše třída je odvozena z QCanvasSprite a nyní má jen konstruktor.

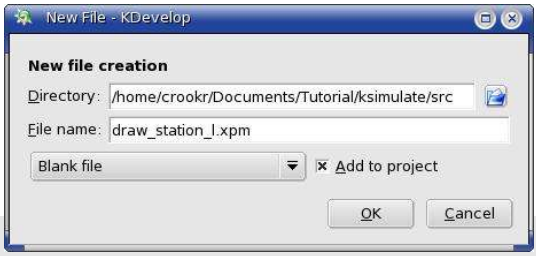
station.cpp

```
#include "station.h"

/*****************************************************************************/
/************************** Station to be simulated **************************/
/*****************************************************************************/

/*************************** constuctor **************************/

Station::Station(QCanvasPixmapArray* sprite, QCanvas* canvas, int x, int y )
                  : QCanvasSprite( sprite, canvas )
{
  // move sprite to correct position and show
  move( x, y );
  show();
}
```

Ke konstruktoru přiřadíme ikonu reprezentující stanici a ukazatel na canvas z konstruktoru QCanvasSprite a potom ikonu umístíme do strávné pozice a pomocí zděděných metod ji zviditelníme.

Obrázek ikony

Obrázek reprezentující stanici na displeji má 16 x 16 pixelů a lze ho vytvořit např. v GIMPu, soubor uložte ve formátu XPM do adresáře. PP: nedělejte to, založíme ho jako text:



draw_station_l.xpm

Tento soubor se do projektu přidá pomocí nabídky File -> „new file“ do adresáře src a do něj textovým editorem napíšeme nížeuvedený kód.

```
/* XPM */
static const char * draw_station_l[] = {
"16 16 2 1",
"       c None",
".      c #000000",
"  ..    ..    ..  ",
"   ..   ..  ..    ",
"    .. .. ..      ",
"     ... ...      ",
"      .. ..       ",
"       ..         ",
"       ..         ",
"   .........      ",
"   .........      ",
"   ..      ..     ",
"   ..      ..     ",
"   ..      ..     ",
"   ..      ..     ",
"   ..      ..     ",
"   .........      ",
"   .........      "};
```

Kreslení na canvas

Stanice zobrazíme na canvas tak, že konstruktorem vytvoříme instanci naší nové třídy "Station" a definujeme souřadnice (x, y). Souřadnice necháme vybrat uživatelem – kliknutí myší na ploše pohledu canvas.

canvasview.h

Implementujeme virtuální chráněnou metodu k zachycení události – levý klik myši – za klauzuli protected.

```
  void contentsMousePressEvent( QMouseEvent* );    // mouse click
```

Přidáme privátní pole pixelů k zápisu do ikony (sprite) – za klauzuli private.

```
  QCanvasPixmapArray    stationSprite;              // station sprite
```

canvasview.cpp

Včleníme station.h pro přístup k funkcionalitě stanice a draw_station_l.xpm pro bitmapový obrázek.

```
#include "station.h"
#include "draw_station_l.xpm"
```

V konstruktoru nastavíme ikonu stanice.

```
  // initialise the station QCanvasPixmapArray
```

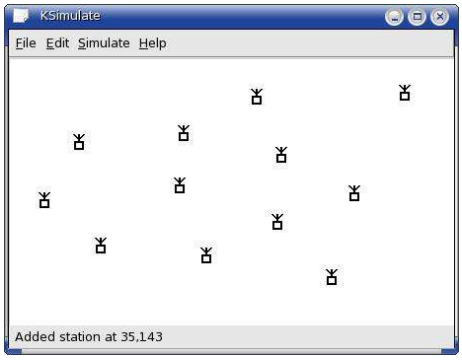
```
  stationSprite.setImage( 0, new QCanvasPixmap( QPixmap(draw_station_1), QPoint(8,8) ) );
```

Vytvoříme metodu contentsMousePressEvent jako novou instanci naší třídy Station, aktualizujeme canvas, zkontrolujeme, zda není zapotřebí kvůli umístění nové stanice zvětšit minimální velikost canvas a nakonec umístíme novou zprávu na stavový řádek.

```
/********************** contentsMousePressEvent ***********************/

void CanvasView::contentsMousePressEvent( QMouseEvent *event )
{
  // add station to canvas
  new Station( &stationSprite, canvas(), event->x(), event->y() );
  canvas()->update();

  // increase minimum canvas size if needed to accomondate new station
  if (event->x()+10 > minCanvasW ) minCanvasW = event->x()+10;
  if (event->y()+10 > minCanvasH ) minCanvasH = event->y()+10;

  // update status bar to say station added
  sbar->message(QString("Added station at %1,%2").arg(event->x()).arg(event->y()));
}
```

Překlad a spuštění kódu

Nový kód bude samočinně zkompilován, jakmile zkusíme znovu spustit aplikaci za pomoci KDevelop. Ověřte chování aplikace.



7. Přidáváme nástrojovou lištu

Základy

V této části chceme implementovat množství změn za účelem editování v aplikaci; 'ADD' pro přidání nové stanice (již implementováno), 'MOVE' pro pohyb stanice do jiné pozice a 'DELETE' pro výmaz stanice. Avšak zatím bychom ještě implementovali funkcionalitu MOVE a DELETE.
Abychom umožnili uživateli přepínat mezi různými editačními módy, přidáme do aplikace nástrojovou lištu se 3 ikonami (jednu pro každý mód) a 3 položky menu do menu edit. Aby uživatel věděl, jaký mód je právě nastaven, zobrazíme nastavený mód jako textové návěští na stavovém řádku.
Funkčnost kódu rozšíří třída Kaction, která obalí 3 uživatelské události a zajistí vstup do editačních módů. Pro komunikaci použijeme poprvé mechanizmu Qt meziobjektové komunikace – signály a sloty.

Aktualizace KSimulate

ksimulate.h

Pro implementaci této funkcionality přidáme veřejnou proměnnou sledující editační mód, veřejnou konstantu pro textové návěští id stavového řádku a 3 sloty pro příjem signálů, které bude uživatel vybírat jako editační módy.

```
  char            editMode;                 // mode, A=ADD, M=MOVE or D=DELETE
  static const int editModeID = 1;          // id of mode on statusbar

public slots:
  void editAdd();                           // set mode to add station
```

```
  void editMove();                              // set mode to move station
  void editDel();                               // set mode to delete station
```

ksimulate.cpp

Včlenění 3 XPM souborů na ikony 3 editačních módů.

```
#include "icon_add.xpm"
#include "icon_move.xpm"
#include "icon_delete.xpm"
```

Včlenění KAction pro přístup k funkcionalitě KAction.

```
#include <kaction.h>
```

V konstruktoru KSimulate přidáme pro každou ze 3 událostí text nabídky, ikonu k zobrazení, klávesovou zkratku, předka slotu, slot volaný touto událostí a interní název. Takto přidáme 3 položky do menu edit a nástrojové lišty. Nástrojová lišta, stejně jako menu a stavový řádek vytvoří samočinně funkcionalita KmainWindow při spuištění aplikace.

```
  // create actions
  KAction *actionAdd  = new KAction( "&Add Station", QIconSet(QPixmap(icon_add_xpm)),
                                     ALT+Key_A, this, SLOT(editAdd()), this, "A");
  KAction *actionMove = new KAction( "&Move Station", QIconSet(QPixmap(icon_move_xpm)),
                                     ALT+Key_M, this, SLOT(editMove()), this, "M");
  KAction *actionDel  = new KAction( "&Delete Station", QIconSet(QPixmap(icon_delete_xpm)),
                                     ALT+Key_D, this, SLOT(editDel()), this, "D");

  // add actions to Edit menu
  actionAdd->plug( menuEdit );
  actionMove->plug( menuEdit );
  actionDel->plug( menuEdit );

  // add actions to tool bar
  actionAdd->plug( toolBar() );
  actionMove->plug( toolBar() );
  actionDel->plug( toolBar() );
```

V konstruktoru KSimulate také musíme inicializovat výchozí editační mód, zakázat události MOVE a DELETE, protože nejsou dosud plně implementovány a aktualizovat stavový řádek, aby zobrazil aktuální mód. Protože ve stavovém řádku používáme textové návěští konstantní šířky, musíme poprvé naplnit text dostatečným množstvím mezer, aby se případné další delší zprávy plně zobrazily.

```
  // initialise the edit mode
  editMode = 'A';
  actionMove->setEnabled(false);
  actionDel->setEnabled(false);
  statusBar()->insertFixedItem( "    ADD    ", editModeID, TRUE );
```

Nakonec přidáme 3 metody slotů pro příjem 3 zpráv od událostí. Každý slot aktualizuje stavový řádek a na daný mód nastaví veřejnou proměnnou.

```
/************************** editAdd **************************/

void KSimulate::editAdd()
{
  statusBar()->message("Edit mode set to ADD");
  editMode = 'A';
  statusBar()->changeItem("ADD", editModeID);
}

/************************** editMove **************************/

void KSimulate::editMove()
{
  statusBar()->message("Edit mode set to MOVE");
  editMode = 'M';
```

```
  statusBar()->changeItem("MOVE", editModeID);
}

/************************** editDel ***************************/

void KSimulate::editDel()
{
  statusBar()->message("Edit mode set to DELETE");
  editMode = 'D';
  statusBar()->changeItem("DELETE", editModeID);
}
```

Přidání 3 ikon

3 ikony zobrazované v nástrojové liště a menu option (???) jsou opět obrázky velikost 16x16 pixelů uložené ve formátu XPM.
Do projektu je přidáme přes File -> new file z Kdevelopu a textovým editorem do nich přidáme následující kód.

icon_add.xpm

```
/* XPM */
static const char * icon_add_xpm[] = {
"16 16 2 1",
"       c None",
".      c #000000",
"  ..    ..    ..  ",
"   ..  ..  ..    ",
"    .. .. ..     ",
"     ... ...     ",
"      .. ..      ",
"       ..        ",
"       ..        ",
"   .........     ",
"   .........     ",
"   ..      ..    ",
"   ..      ..    ",
"   ..      ..    ",
"   ..      ..    ",
"   ..      ..    ",
"   .........     ",
"   ........      "};
```

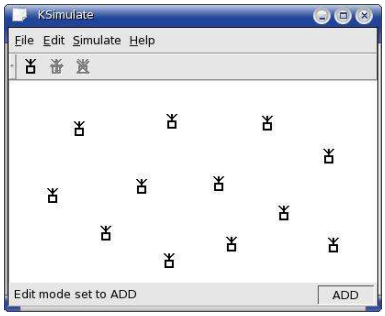
icon_move.xpm

```
/* XPM */
static const char * icon_move_xpm[] = {
"16 16 3 1",
"       c None",
".      c #000000",
"+      c #0015FF",
"  ..    .+    ..  ",
"   ..  +++ ..    ",
"    ..+++++.     ",
"     ..+ ++.     ",
"      .+ ++      ",
"   +    +++   +  ",
"  +++++++++++++  ",
" +++++++++++++++ ",
"  +++++++++++++  ",
"   +.  +++ ..+   ",
"   ..  +++ ..    ",
"   ..  +++ ..    ",
"   .. +++ ..     ",
"   ..  +++ ..    ",
"   .....+.. ..   ",
"   ........ ..   "};
```

icon_delete.xpm

```
/* XPM */
static const char * icon_delete_xpm[] = {
"16 16 3 1",
"        c None",
".       c #000000",
"+       c #FF0000",
"   ..    ..    ..   ",
"    ..   ..   ..    ",
" ++  ..  ..  .. ++ ",
" ++ + ... ...  ++ + ",
"  ++ + .. .. ++ +  ",
"   +++ ..  +++     ",
"    ++ +. .++ +    ",
"   . .+++ +++ ..   ",
"   . ..++ ++. ..   ",
"   . . ++ ++  ..   ",
"   . .+++ +++ ..   ",
"   .+++   ++ +.    ",
"   +++      +++    ",
"  ++ +       ++ +  ",
" ++ +.. .. ... .+ ++ ",
" ++ ... .. ... .. ++ "};
```

Překlad a spuštění kódu



Nový kód bude samočinně zkompilován, jakmile zkusíme znovu spustit aplikaci za pomoci Kdevelop. Ověřte chování aplikace, když zakomentujete v konstruktoru Ksimulate příkaz setEnabled(false) (?).

8. Přesun a mazání položek canvas

Základy

Nyní implementujeme funkcionalitu editačních módů 'MOVE' a 'DELETE'. Zadání módu 'MOVE' (menu, zkratková klávesa, nástrojová lišta) bude uživatel schopen pohybovat stanicemi tak, že je myší vybere a přetáhne do nové pozice na canvas. Podobně po spuštění módu 'DELETE' může uživatel kliknutím myši smazat stanici.

Aktualizování kódu

ksimulate.cpp

V konstruktoru KSimulate musíme odstranit 2 řádky zabraňující realizaci událostí obalujících módy MOVE a DELETE.

```
  actionMove->setEnabled(false);
  actionDel->setEnabled(false);
```

canvasview.h

Implementujeme virtuální chráněnou (protected) metodu zachycující události pohybu myši.

```
  void contentsMouseMoveEvent( QMouseEvent* );      // mouse move
```

Přidáme privátní ukazatel Add a private pointer pro zajištění vhodného přístupu do veřejné proměnné a privátní proměnnou zachycující trasu pohybu stanice.

```
KSimulate*          ksim;                   // shortcut to KSimulate
QCanvasItem*        moving;                 // pointer to station being moved
```

canvasview.cpp

V konstruktoru inicializujeme ukazatel na KSimulate.

```
ksim = parent;
```

V metodě contentsMouseMoveEvent kontrolujeme, zda je stanice právě přesouvána a zda je nutno ji na canvas překreslit podle polohy myši. Canvas musíme aktualizovat po každé změně, jinak novou pozici neuvidíme, dokud nebude canvas aktualizováno z nějakého jiného důvodu.

```
/********************** contentsMouseMoveEvent **********************/

void CanvasView::contentsMouseMoveEvent( QMouseEvent *event )
{
  // if a station is being moved, move to new position
  if ( moving )
  {
    // ensure new position is still on canvas
    int x = event->x();
    int y = event->y();
    if ( x < 10 ) x = 10;
    if ( y < 10 ) y = 10;
    if ( x+10 > canvas()->width()  ) x = canvas()->width()  - 10;
    if ( y+10 > canvas()->height() ) y = canvas()->height() - 10;

    // move station to new position
    moving->move( x, y );
    canvas()->update();
  }
}
```

Potřebujeme také aktualizovat metodu contentsMousePressEvent, aby se chovala v každém editačním módu jinak. Nížeuvedený kód používá 3 příkazy if, pro každý editační mód jeden, ale stejně snadno by se dal použít příkaz switch.

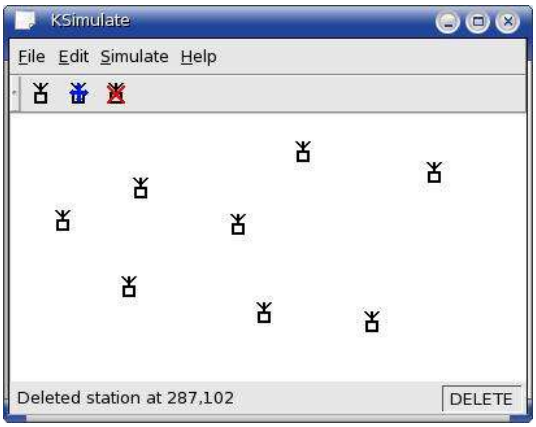
První věcí, kterou musíme udělat, je inicializace 'pohyblivého' ukazatele, abychom zajistili, že contentsMouseMoveEvent nikdy nezkusí pohybovat špatnou, nebo neplatnou stanicí. V módu 'ADD' najdete většinu kódu z dřívějška. Kód pro 'MOVE' i 'DELETE' začíná získáním ukazatele na stanici odpovídající ukazateli uživatelského výběru na canvas pomocí funkcionality "collisions()". Pokud vrácený seznam není prázdný, identifikovali jsme 1 nebo více stanic k přesunu/smazání.

```
// initialise station moving pointer each time mouse clicked
moving = 0;

if (ksim->editMode == 'A')   // if edit mode is ADD
{
  int x = event->x();
  int y = event->y();
  if ( x < 10 ) x = 10;
  if ( y < 10 ) y = 10;

  // add station to canvas
  new Station( &stationSprite, canvas(), x, y );
  canvas()->update();

  // increase minimum canvas size if needed to accomondate new station
  if ( x+10 > minCanvasW ) minCanvasW = x+10;
  if ( y+10 > minCanvasH ) minCanvasH = y+10;

  // update status bar to say station added
```

```
    sbar->message(QString("Added station at %1,%2").arg(x).arg(y));
  }

  if (ksim->editMode == 'M')    // if edit mode is MOVE
  {
    // find station to be moved
    QCanvasItemList  list = canvas()->collisions( event->pos() );
    if (!list.isEmpty())
    {
      moving = *list.begin();
      sbar->message(QString("Moving station from %1,%2").arg(event->x()).arg(event->y()));
    }
  }

  if (ksim->editMode == 'D')    // if edit mode is DELETE
  {
    // find station to be deleted
    QCanvasItemList  list = canvas()->collisions( event->pos() );
    if (!list.isEmpty())
    {
      delete *list.begin();
      canvas()->update();
      sbar->message(QString("Deleted station at %1,%2").arg(event->x()).arg(event->y()));
    }
  }
```

Překlad a spuštění kódu

Nový kód bude samočinně zkompilován, jakmile zkusíme znovu spustit aplikaci za pomoci KDevelop. Ověřte chování aplikace.



9. Ukládání do souboru

Ukládání do XML souboru

Nyní dovolíme uživateli uložit simulovaná data do souboru formátu XML. Přidáme také nabídku option, ikonu na nástrojovou lištu a použijeme funkcionalitu KfileDialog pro výběr jména souboru a umístění.

Aktualizace KSimulate

Ve třídě KSimulate přidáme nový slot "fileSave" přiřazený k nové akci/operaci/činnosti (?) "actionSave". Přidáme také privátní ukazatel odkazující na CanvasView zevně konstruktoru.

ksimulate.h

Raději, než začlenění hlavičkového souboru použij pro zjednodušení a zrychlení překladu dopřednou deklaraci pro třídu CanvasView.

```
class CanvasView;
```

Přidej veřejný (public) slot pro příjem signálu, že uživatel chce ukládat.

```
  void fileSave();                          // save current simulation
```

Definuj privátní ukazatel odkazující na CanvasView.

```
private:
  CanvasView*      canvasView;                  // pointer to canvas view
```

ksimulate.cpp

Přidej hlavičkové soubory zpřístupňující funkcionalitu KFileDialog, KUser, QDom, a QdateTime.

```
#include <kfiledialog.h>
#include <kuser.h>
#include <qdom.h>
#include <qdatetime.h>
```

V konstruktoru vytvoř novou akci zapouzdřující zápis do souboru pomocí standardní akce poskytované KStdAction.

```
  KAction *actionSave = KStdAction::save(this, SLOT(fileSave()), 0);
```

Přidej novou položku do nabídky File.

```
  // add actions to File menu
  actionSave->plug( menuFile );
```

A přidej novou položku (akci ?) do nástrojové lišty.

```
  actionSave->plug( toolBar() );
```

Modifikuj příkaz vytvářející CanvasView k použití privátní proměnné Ksimulate namísto lokální proměnné.

```
  canvasView = new CanvasView( canvas, this );
```

Nakonec přidej metodu slotu zachycující signál a ukládající simulovaná data do .XML souboru. Jak je uvedeno výše, použijeme k tomu KfileDialog – okno pro zadání resp. výběr jména souboru. Soubor XML sestavíme v paměti pomocí funkcionality Qdom a potom ho zapíšeme do souboru jako proud textu (text stream ?). Funkce vrací true pokud je uložení úspěšné, jinak vrací false.

```
/********************************* fileSave **********************************/

bool KSimulate::fileSave()
{
  // get user to select filename and location
  QString fileName = KFileDialog::getSaveFileName();
  if (fileName.isEmpty()) return false;

  // start the XML document
  QDomDocument doc( "KSimulate" );

  // create the main data element and station elements
  QDomElement data = doc.createElement( "data" );
  KUser user(KUser::UseRealUserID);
  data.setAttribute( "savedBy", user.loginName() );
  data.setAttribute( "savedAt", QDateTime::currentDateTime().toString(Qt::LocalDate) );
  doc.appendChild( data );
  canvasView->saveStations( &doc, &data );

  // open the file and check we can write to it
  QFile file(fileName);
  if (!file.open(IO_WriteOnly))
  {
    statusBar()->message(QString("Failed to save '%1'").arg(fileName));
    return false;
  }

  // output the XML document as a text stream to the file
  QTextStream ts(&file);
  ts << doc.toString();
  file.close();
```

```
  statusBar()->message(QString("Saved to '%1'").arg(fileName));
  return true;
}
```

Aktualizace CanvasView

canvasview.h

Pro podporu slotu fileSave, v Ksimulate přidáme novou veřejnou (public) metodu CanvasView pro přidání stanice do XML dokumentu.

```
  void saveStations( QDomDocument*, QDomElement* );     // save stations to XML document
```

canvasview.cpp

Novou metodou vložíme XML prvek pro každou stanici včetně X a Y souřadnic.

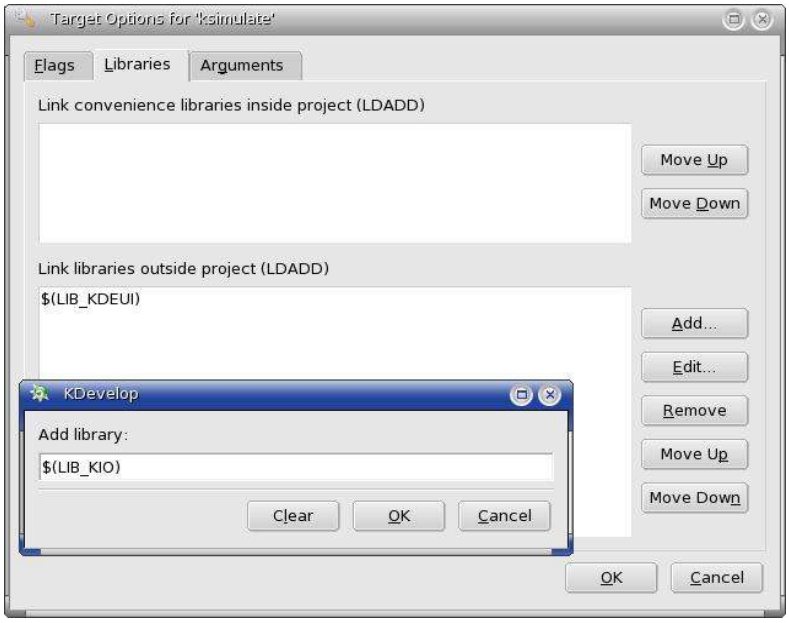
```
/********************************* saveStations *********************************/

void CanvasView::saveStations(QDomDocument* doc, QDomElement* data)
{
  // get list of all stations and add to XML document data element
  QCanvasItemList  list = canvas()->allItems();
  for( QCanvasItemList::iterator it = list.begin(); it != list.end(); ++it )
  {
    QDomElement stn = doc->createElement( "station" );
    stn.setAttribute( "x", (*it)->x() );
    stn.setAttribute( "y", (*it)->y() );
    data->appendChild( stn );
  }
}
```

Zároveň využijeme příležitosti k odstranění zprávy ze stavového řádku pokaždé, když změníme velikost canvas v "viewportResizeEvent".

```
  // update status bar to say resize happened
  sbar->message(QString("Canvas resized to %1, %2").arg(w).arg(h));
```

Linkování s novou knihovnou

Protože třídu "KfileDialog" vezmeme namísto ze standardní knihovny KDEUI z knihovny KIO – to musíme říci KDevelopu, aby při linkování naší aplikace použil zvláštní knihovnu. To se provádí v Automake Manageru, který otevřeme z ikony na pravé straně KDevelopu. Otevřete Automake Manager, klikněte na ikonu "Show options" (vypadá, jako montážní klíč). Vyber záložku Libraries, stiskněte čudlík Add a zadejte "$(LIB_KIO)". Zadejte OK a OK.



Překlad a spuštění kódu

Nový kód bude samočinně zkompilován, jakmile zkusíme znovu spustit aplikaci za pomoci KDevelop. Ověřte chování aplikace.

10. Načtení souboru

Načtení XML souboru

Nyní dovolíme uživateli nahrát simulaci z dříve uloženého souboru ve formátu XML. Přidáme položku nabídky a ikonu do nástrojové lišty a za pomoci funkcionality KfileDialog dovolíme uživateli vybrat jméno souboru a umístění.

Aktualizace KSimulate

Ve třídě Ksimulate vytvoříme nový slot "fileOpen" přiřazený nové akci "actionOpen".

ksimulate.h

Přidej veřejný slot pro příjem signálu požadavku otevření souboru.

```
  bool fileOpen();                              // open previously saved simulation
```

ksimulate.cpp

V konstruktoru vytvoříme novou akci zapouzdřující otevření souboru pomocí standardní funkce poskytované KStdAction.

```
  KAction *actionOpen = KStdAction::open(this, SLOT(fileOpen()), 0);
```

Přidej novou akci do nabídky File.

```
  actionOpen->plug( menuFile );
```

A přidej novou akci k nástrojové liště.

```
  actionOpen->plug( toolBar() );
```

Nakonec přidej metodu slotu pro příjem signálu a natažení simulačních dat ze souboru XML. K tomu použijeme KFileDialog – uživatel si vybere soubor a umístění. Potom natáhneme XML dokument do paměti pomocí funkcionality QDOM a nakonec data zpracujeme a vytvoříme nové stanice. Funkce vrací true pokud je uložení úspěšné, jinak vrací false.

```
/********************************* fileOpen ***********************************/

bool KSimulate::fileOpen()
{
  // get user to select filename and location
  QString fileName = KFileDialog::getOpenFileName();
  if (fileName.isEmpty()) return false;

  // create an empty XML document
  QDomDocument doc( "KSimulate" );

  // open the file and check we can read from it
  QFile file(fileName);
  if (!file.open(IO_ReadOnly))
  {
    statusBar()->message(QString("Failed to open '%1'").arg(fileName));
```

```
    return false;
  }

  // read the XML document from the file
  if (!doc.setContent(&file))
  {
    file.close();
    statusBar()->message(QString("Failed to read '%1'").arg(fileName));
    return false;
  }
  file.close();

  // check the document type is correct
  if (doc.doctype().name() != "KSimulate")
  {
    statusBar()->message(QString("Invalid '%1'").arg(fileName));
    return false;
  }

  // read the XML elements to create the stations
  QDomElement data = doc.documentElement();
  canvasView->loadStations( &data );
  statusBar()->message(QString("Loaded '%1'").arg(fileName));
  return true;
}
```

Aktualizace CanvasView

canvasview.h

Pro podporu slotu Ksimulate fileOpen vytvoříme novou metodu CanvasView čtoucí XML dokument a vytvářející stanice.

```
  void loadStations( QDomElement* );                    // create stations from XML
```

canvasview.cpp

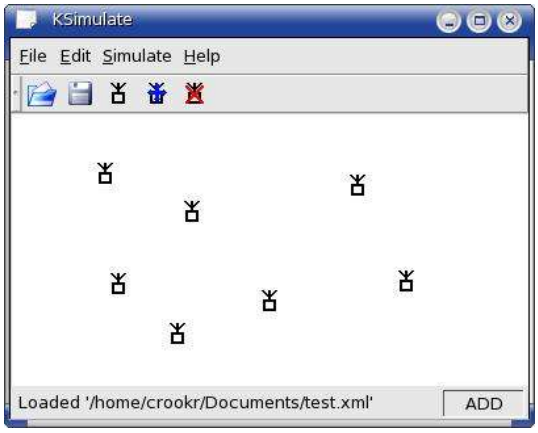
V nové metodě začneme smazáním starých stanic a resetováním minimální velikost canvas a potom přečteme XML dokument. Pro každý prvek stanice vytvoříme novou stanici v příslušné pozici a minimální velikost canvas (?). Nakonec změníme velikost canvas a překreslíme okno.

```
/********************************* openStations *********************************/

void CanvasView::loadStations( QDomElement* data )
{
  // delete existing stations
  QCanvasItemList  list = canvas()->allItems();
  for( QCanvasItemList::iterator it = list.begin(); it != list.end(); ++it )
    delete *it;

  // reset minimum canvas size
  minCanvasW = MIN_CANVAS_W;
  minCanvasH = MIN_CANVAS_H;

  // read XML document and create new stations
  for ( QDomNode node = data->firstChild(); !node.isNull(); node = node.nextSibling() )
    {
      QDomElement stn = node.toElement();
      int         x   = stn.attribute("x").toInt();
      int         y   = stn.attribute("y").toInt();

      // ensure station position and minimum canvas size updated as necessary
      if ( x < 10 ) x = 10;
      if ( y < 10 ) y = 10;
      if ( x+10 > minCanvasW ) minCanvasW = x+10;
      if ( y+10 > minCanvasH ) minCanvasH = y+10;

      new Station( &stationSprite, canvas(), x, y );
```

```
    }

  // resize and update canvas so new stations are displayed
  int w = visibleWidth();
  int h = visibleHeight();
  if ( w < minCanvasW ) w = minCanvasW;
  if ( h < minCanvasH ) h = minCanvasH;
  canvas()->resize( w, h );
  canvas()->update();
}
```

Překlad a spuštění kódu

Nový kód bude samočinně zkompilován, jakmile zkusíme znovu spustit aplikaci za pomoci Kdevelop. Ověřte chování aplikace.



11. Nový soubor a konec

Přidání 2 nových položek nabídky

V této části přidáme 2 položky do nabídky File – jednu pro spuštění nové simulace a druhou pro ukončení aplikace. Na nástrojovou lištu přidáme tlačítko pro novou simulaci. Abychom pomohli uživateli zabránit ztrátě práce, budou před vlastní činností obě volby nabízet uložení stávající simulace.

Aktualizace kódu

ksimulate.h

Ve třídě KSimulate implementujeme virtuální funkci "queryClose"volané při požadavku ukončení aplikace.

```
protected:
  bool queryClose();                          // check user happy to quit
```

A přidáme veřejný slot pro příjem signálu – požadavku nové simulace.

```
  bool fileNew();                             // start new simulation
```

ksimulate.cpp

Včleníme hlavičkový soubor zpřístupňující funkcionalitu KmessageBox.

```
#include <kmessagebox.h>
```

V konstruktoru vytvoříme 2 nové akce zapouzdřující funkce New a Edit za pomoci standardní definice funkce poskytované KstdAction. Funkce „New“ je asociována s naším novým veřejným slotem; funkce „Quit“ je spojena se zděděným slotem QWidget "close".

```
  KAction *actionNew  = KStdAction::openNew(this, SLOT(fileNew()), 0);
  KAction *actionQuit = KStdAction::quit(this, SLOT(close()), 0);
```

Přidáme položky New a Quit do nabídky File – okolo existujících položek a včetně oddělovače k oddělení položky Quit od ostatních.

```
  actionNew->plug( menuFile );
  actionOpen->plug( menuFile );
  actionSave->plug( menuFile );
  menuFile->insertSeparator();
  actionQuit->plug( menuFile );
```

A přidáme funkci New do nástrojové lišty.

```
  actionNew->plug( toolBar() );
```

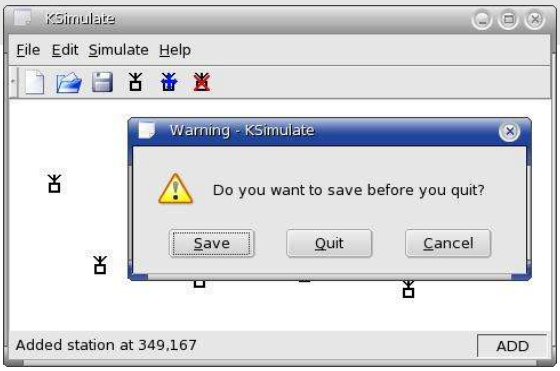
Nyní můžeme přidat kód pro funkcionalitu "queryClose". Pokud nejsou žádné stanice, provedeme ukončení okamžitě, jinak pomocí KMessageBox dáme uživateli možnost uložit simulaci na disk před ukončením aplikace.

```
/*********************************** queryClose ***********************************/

bool KSimulate::queryClose()
{
  // no need to do anything if no stations
  QCanvasItemList list = canvasView->canvas()->allItems();
  if (list.isEmpty()) return true;

  // check if user wants to save before quitting
  while (true)
    switch (KMessageBox::warningYesNoCancel(this,
        "Do you want to save before you quit?",
        QString::null, QString("&Save"), QString("&Quit")) )
    {
      case KMessageBox::Yes:  // "Save"
        // if save not successful ask again
        if (!fileSave()) break;

      case KMessageBox::No:   // "Quit"
        return true;

      default:                // "Cancel"
        return false;
    }
}
```

Podobně přidáme kód pro slot spuštění nové simulace. Pokud není žádná stanice, provedeme ihned. Jinak pomocí KMessageBox dáme uživateli možnost uložit simulaci na disk před výmazem stanic a aktualizací canvas.

```
/************************************ fileNew *************************************/

bool KSimulate::fileNew()
{
  // no need to do anything if no stations
  QCanvasItemList list = canvasView->canvas()->allItems();
  if (list.isEmpty()) return true;

  // check if user wants to save before starting new simulation
  while (true)
    switch (KMessageBox::warningYesNoCancel(this,
        "Do you want to save before you start a new simulation?",
        QString::null, QString("&Save"), QString("&New")) )
    {
      case KMessageBox::Yes:  // "Save"
        // if save not successful ask again
        if (!fileSave()) break;

      case KMessageBox::No:   // "New"
        // delete every station
        for( QCanvasItemList::iterator it = list.begin(); it != list.end(); ++it )
          delete *it;
        canvasView->canvas()->update();
        statusBar()->message("New simulation started");
```

```
        return true;

      default:                  // "Cancel"
        return false;
    }
}
```

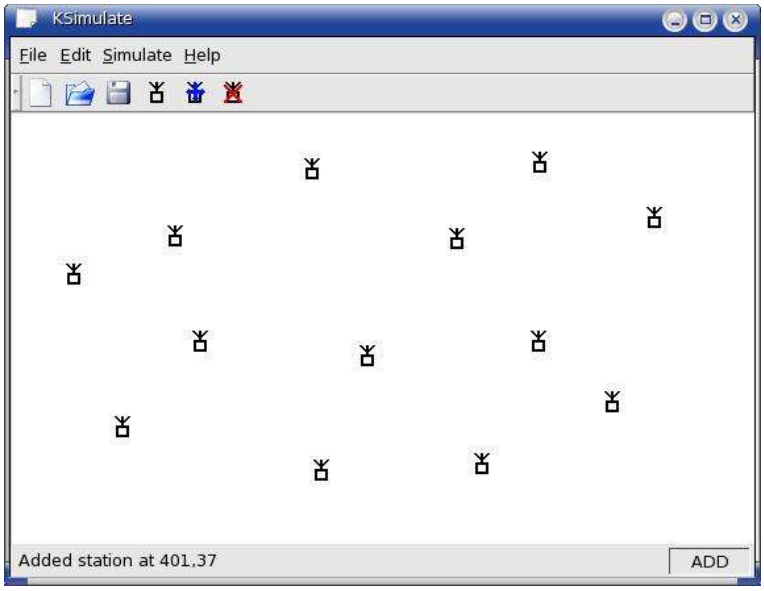


Překlad a spuštění kódu

Nový kód bude samočinně zkompilován, jakmile zkusíme znovu spustit aplikaci za pomoci KDevelop. Ověřte chování aplikace.

12. Závěr

Tutoriál pro KDE programování



Doufám, že jste si vychutnali tento tutoriál a seznámili se se snadností programování pomocí KDevelop, KDE a Qt. Toto je však jen náčrt možností. Prosím kontaktujte mě s připomínkami a návrhy ke zlepšení tutoriálu. Překladatel se obrací s prosbou případného upřesnění překladu a terminologie.

Poznámky překladatele

KDevelop se snaží být chytrý – pokud vložíme kód na nesprávné místo a porušuje takto přímo syntaxi, objeví se vlevo červený křížek na znamení chyby.

Zdrojáky viz http://www.dazzle.plus.com/linux/index.htm