Multiplatformní GUI toolkity

GTK+ (gtkmm), Qt a wxWidgets

Jan Outrata

březen 2006 (aktualizace duben 2009)

Historie

X Window System (X, X Windows)

- standardní grafické uživatelské rozhraní (GUI) původně pro UNIXové systémy
- oddělený od OS, síťový (X klient/server), otevřený (X.org), vrstevná modulární architektura (ovladače, X protokol, Xlib, toolkity, aplikace, WM)

Xlib

- API funkcí X na zprávy X protokolu (okna, vstup)
- neposkytuje prvky GUI, jen funkce pro jejich vytvoření
- složité a nevhodné pro přímé použití pro tvorbu GUI $\rightarrow$ toolkity

Toolkity

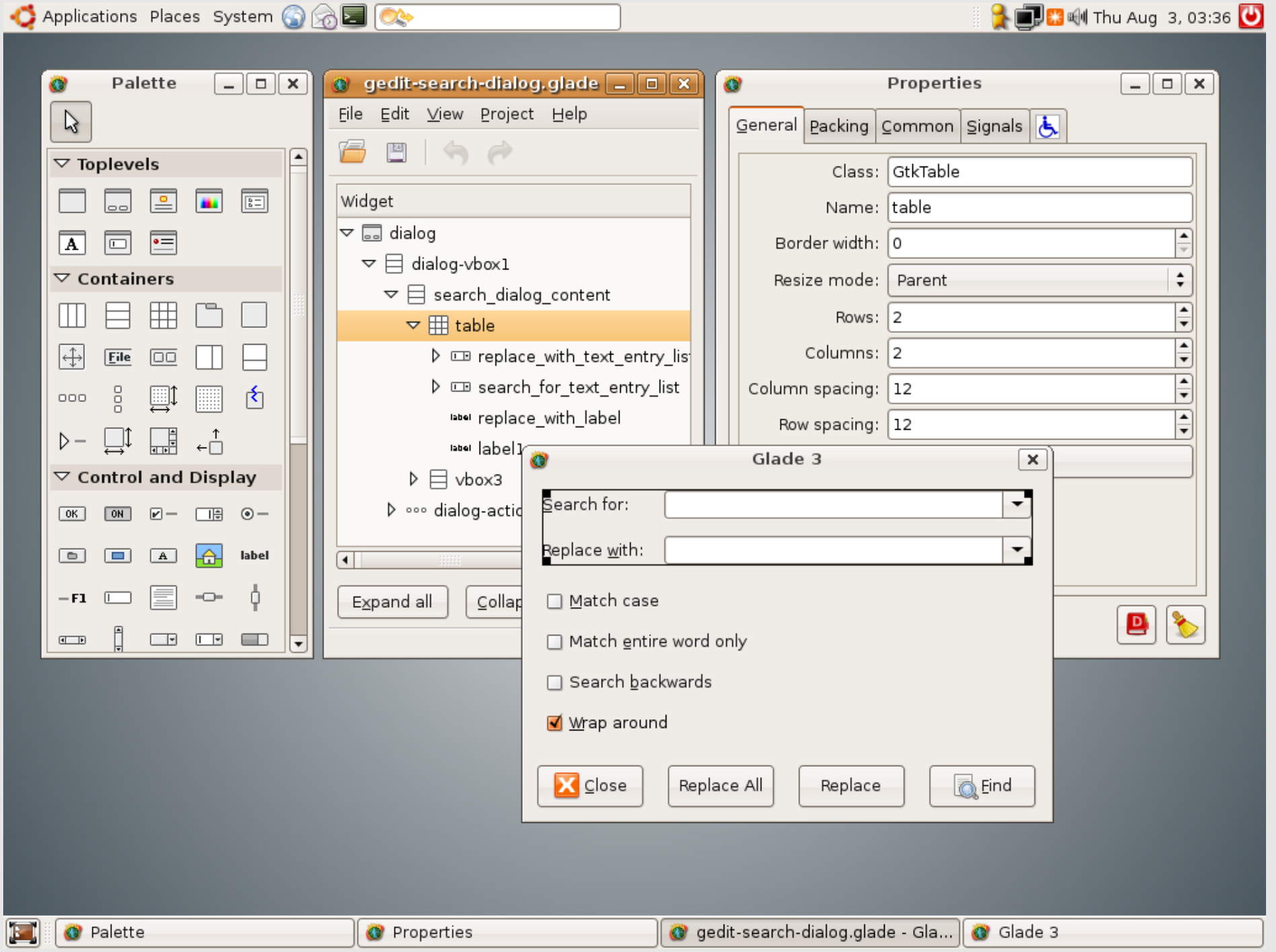
- poskytují prvky GUI (tlačítka, menu, . . .)
- událostmi řízené programování (hlavní smyčka, obsluhy událostí)
- objektově orientované programování (objekty = prvky GUI) - nevyžaduje OO programovací jazyk!
- thread-safe - kód toolkitu lze mít ve více vláknech

GTK+ (The GIMP Toolkit, http://www.gtk.org)

- napsaný v C, varianty API (language bindings)pro C++ (gtkmm, dříve Gtk--, http://www.gtkmm.org), C#/.NET (Gtk#/.NET, projekt Mono, http://gtk-sharp.sourceforge.net/), Javu, Perl, Python, Ruby, Common Lisp a další
- vznikl jako toolkit pro GIMP (podle Motifu) kolem roku 1996
- multiplatformní, pro UNIXové systémy, M$ Windows, Mac OS - jednotný vzhled, pro embedded Linux GMAE (GNOME Mobile & Embedded)
- **Glade** (**Glademm**) - nástroj pro grafický návrh částí GUI, z XML generuje kostru, také dynamicky vytvářené GUI za běhu pomocí knihovny libglade (libglademm)
- dokumentace, referenční manuál, tutoriály, knihy
- nejnovější (stabilní) verze 2.16.x
- licence GNU LGPL
- v gtkmm omezení při práci s výjimkami (C nezná výjimky), využívá STL

- nad ním **GNOME** - přidává abstrakci soubového systému (GIO a GVFS), uživatelská nastavení (GConf), komunikaci aplikací (D-Bus), komponenty a vzdálené volání (Bonobo), desktop (typy souborů, panel, aplety, souborový a okenní manažer, hesla, sezení, multimedia, tisk, . . .), office (GNOME Office)

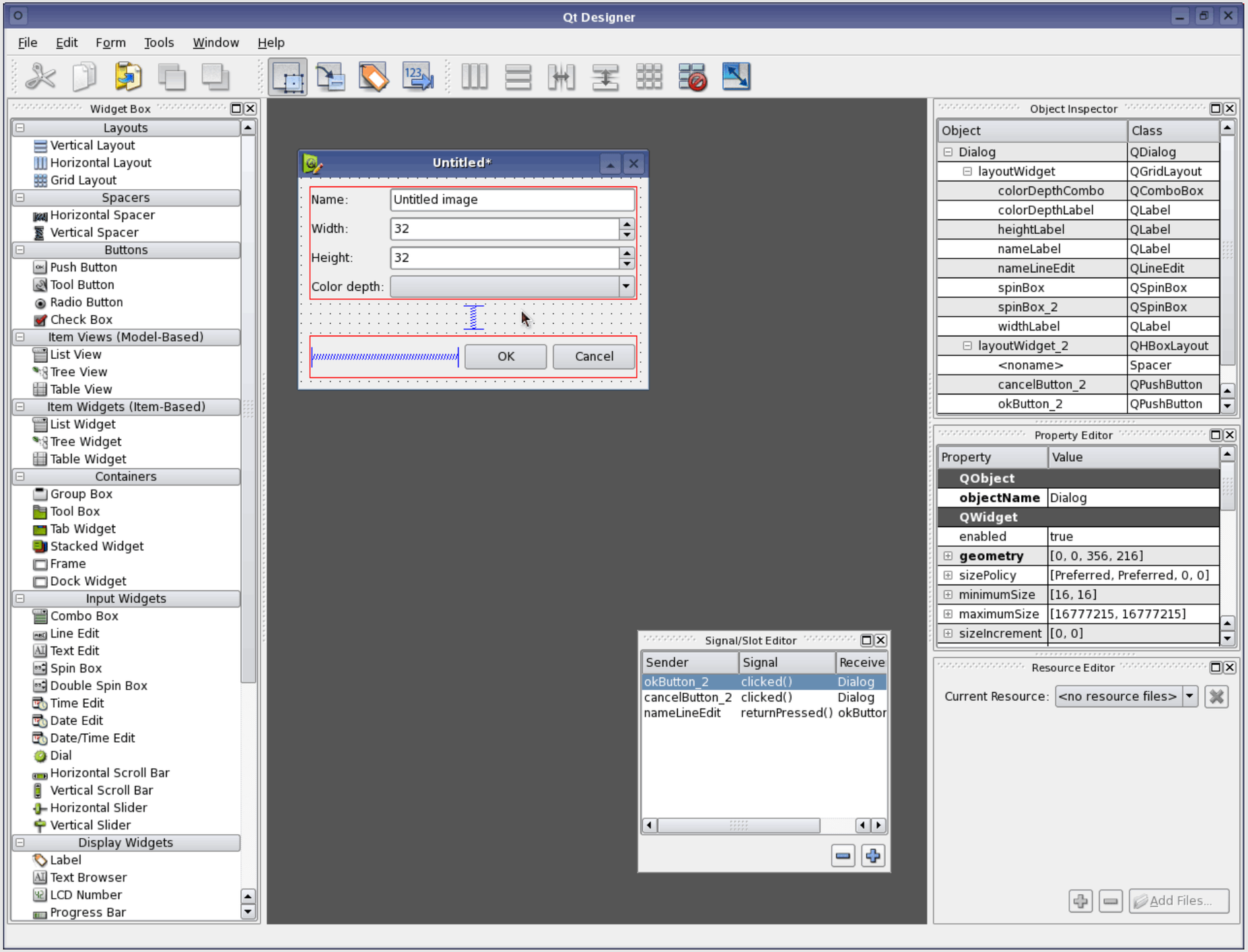
- GNOME Human Interface Guidelines (http://library.gnome.org/devel/hig-book/stable/)


Applications Places System
Thu Aug 3, 03:36
Palette
Toplevels
Containers
File
Control and Display
OK
ON
label
F1
gedit-search-dialog.glade
File Edit View Project Help
Widget
dialog
dialog-vbox1
search_dialog_content
table
replace_with_text_entry_lis
search_for_text_entry_list
replace_with_label
label1
vbox3
dialog-actic
Expand all
Collap
Properties
General Packing Common Signals
Class: GtkTable
Name: table
Border width: 0
Resize mode: Parent
Rows: 2
Columns: 2
Column spacing: 12
Row spacing: 12
Glade 3
Search for:
Replace with:
Match case
Match entire word only
Search backwards
Wrap around
Close
Replace All
Replace
Find
Palette
Properties
gedit-search-dialog.glade - Gla...
Glade 3

Qt (http://www.qtsoftware.com/)

- napsaný v C++, varianty API (language bindings) pro C#/.NET (projekt Mono), Ruby, Perl, Python a další (Java končí)
- původní firemní řešení toolkitu z roku 1995 (původně firma Trolltech, nyní Qt Software)
- multiplatformní, pro UNIXové systémy, M$ Windows, Mac OS - nativní vzhled (styly), pro embedded Linux Qt Extended (dříve Qtopia)
- **Qt Designer** - nástroj pro grafický návrh částí GUI, XML výstup (*.ui) se do C++ překládá pomocí `uic` (User Interface Compiler), také dynamicky vytvářené GUI za běhu pomocí *QWidgetFactory* + třída pro sloty
- **Qt Linguist** - nástroj pro podporu překladu řetězců v programu (lokalizace)
- **Qt Assistant** - nástroj pro podporu tvorby dokumentace a nápovědy
- dokumentace, referenční manuál, tutoriál, whitepaper, knihy
- nejnovejší (stabilní) verze 4.5.x

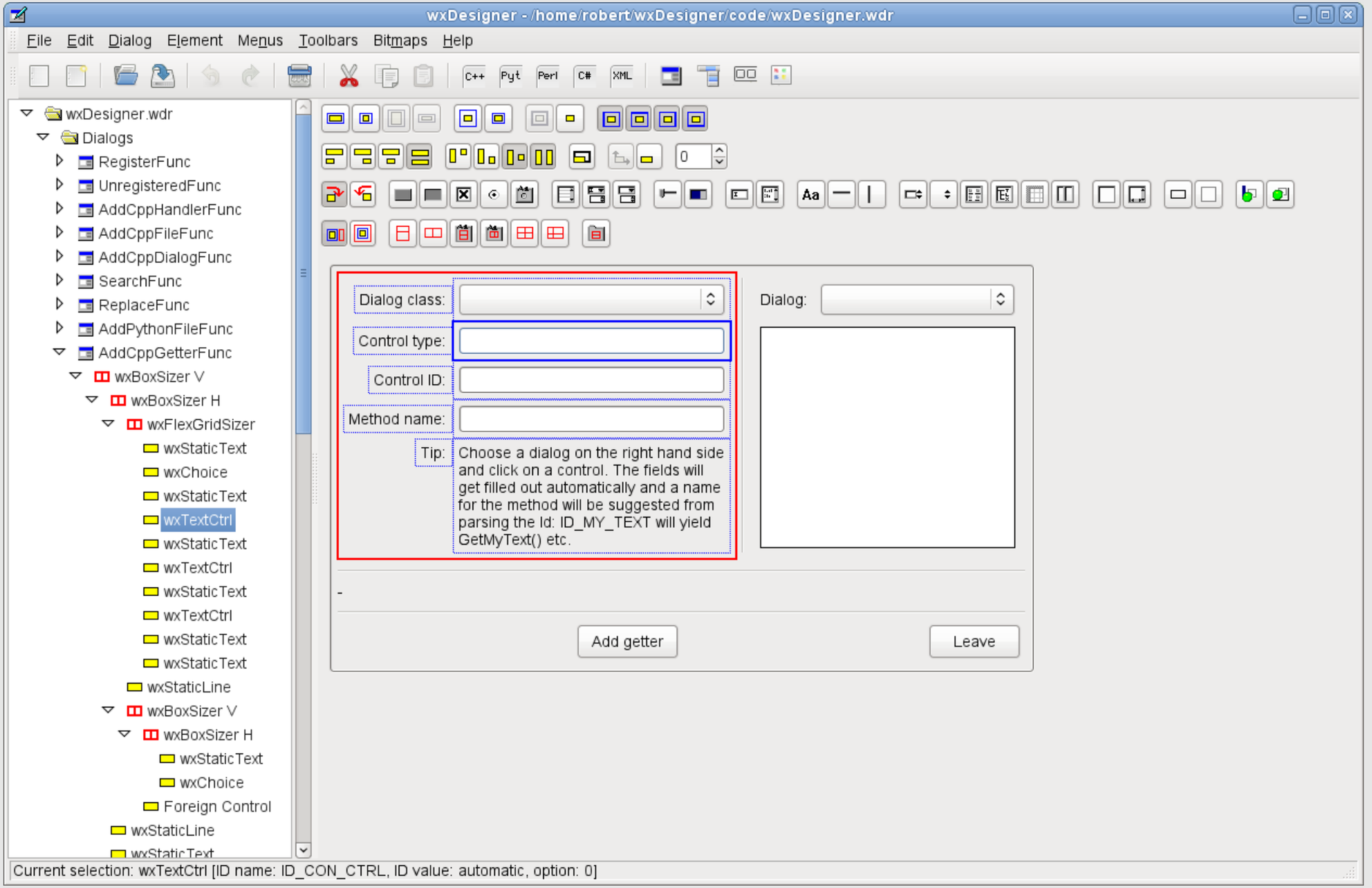
- licence GNU LGPL/GPL a Qt Commercial (30 dní vyzkoušení)
- rozšiřuje C++ o komunikaci mezi objekty (signály a sloty) a dynamickou identifikaci typů (pro každou třídu existuje meta-objekt, Meta Object System) - pro překlad do C++ je `moc` (Meta Object Compiler)
- nad ním **KDE** - neblokující I/O (KIO), HW abstrakci (Solid, kioslaves), multimedia (Phonon), komunikaci (Decibel, D-Bus), komponenty (KParts), desktop (Plasma, KWin, panel, aplety, session, . . .), office (KOffice)
- KDE Human Interface Guidelines (http://usability.kde.org/hig/)

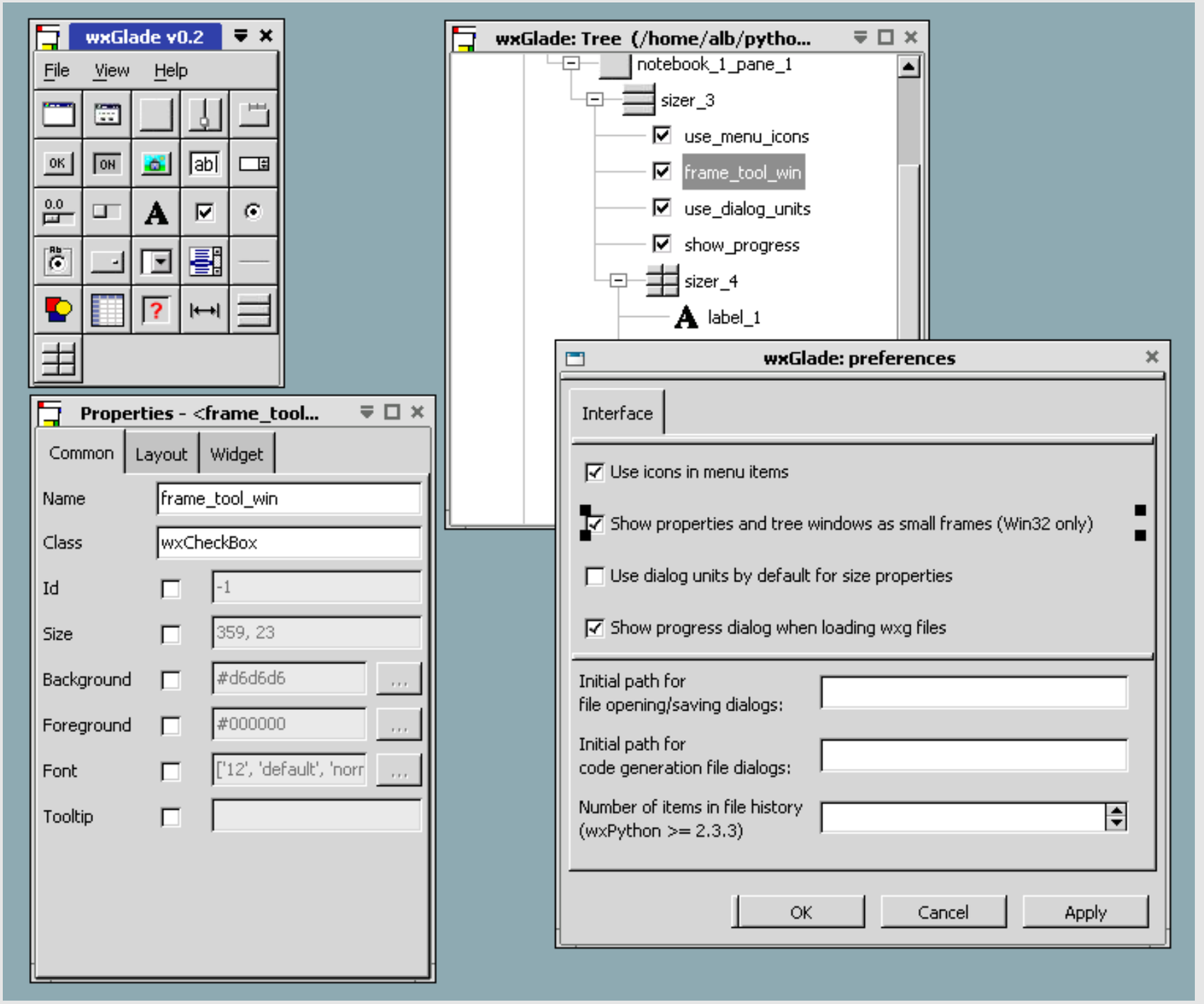

Qt Designer
File Edit Form Tools Window Help
Widget Box
Layouts
Vertical Layout
Horizontal Layout
Grid Layout
Spacers
Horizontal Spacer
Vertical Spacer
Buttons
Push Button
Tool Button
Radio Button
Check Box
Item Views (Model-Based)
List View
Tree View
Table View
Item Widgets (Item-Based)
List Widget
Tree Widget
Table Widget
Containers
Group Box
Tool Box
Tab Widget
Stacked Widget
Frame
Dock Widget
Input Widgets
Combo Box
Line Edit
Text Edit
Spin Box
Double Spin Box
Time Edit
Date Edit
Date/Time Edit
Dial
Horizontal Scroll Bar
Vertical Scroll Bar
Horizontal Slider
Vertical Slider
Display Widgets
Label
Text Browser
LCD Number
Progress Bar
Untitled*
Name:
Untitled image
Width:
32
Height:
32
Color depth:
OK
Cancel
Object Inspector
Object
Class
Dialog
QDialog
layoutWidget
QGridLayout
colorDepthCombo
QComboBox
colorDepthLabel
QLabel
heightLabel
QLabel
nameLabel
QLabel
nameLineEdit
QLineEdit
spinBox
QSpinBox
spinBox_2
QSpinBox
widthLabel
QLabel
layoutWidget_2
QHBoxLayout
<noname>
Spacer
cancelButton_2
QPushButton
okButton_2
QPushButton
Property Editor
Property
Value
QObject
objectName
Dialog
QWidget
enabled
true
geometry
[0, 0, 356, 216]
sizePolicy
[Preferred, Preferred, 0, 0]
minimumSize
[16, 16]
maximumSize
[16777215, 16777215]
sizeIncrement
[0, 0]
Signal/Slot Editor
Sender
Signal
Receive
okButton_2
clicked()
Dialog
cancelButton_2
clicked()
Dialog
nameLineEdit
returnPressed()
okButtor
Resource Editor
Current Resource:
<no resource files>
Add Files...

wxWidgets (Windows-X-Widgets, wxWindows, http://www.wxwidgets.org)

- napsaný v C++, varianty API (language bindings) pro C#/.NET (wx.NET, http://wxnet.sourceforge.net/ , Perl, Python, Basic
- vznikl jako multiplatformní toolkit nad MFC 1.0 (později Windows API) a XView (později Motif a GTK+) na Univerzitě v Edinburgu v roce 1992
- multiplatformní (wxUniversal), pro UNIXové systémy (wxGTK, wxX11), M$ Windows (wxMSW), Mac OS (wxMac) - nativní vzhled i prvky (využívá jiné knihovny - GTK+, Xlib, WIN32, Carbon), pro embedded wxEmbedded
- více orientované na M$ Windows MFC (speciální objekty, systém událostí podobný MFC message maps) - jednodušší portace z M$ Windows
- **wxDesigner**, **wxGlade** - nástroje pro grafický návrh částí GUI (resource XML), také dynamicky vytvářené GUI za běhu pomocí knihovny libwxxrc
- nástroje na resource soubory (`convertrc`, `wxrc`, `XRCed`)
- dokumentace, referenční manuál, tutoriály, kniha

- nejnovejší (stabilní) verze 2.8.x
- licence wxWindows Licence (GNU LGPL + 1 odstavec o libovolném způsobu distribude binárních odvozených děl)
- používá podmnožinu C++ z důvodu portability - ne výjimky, šablony (má pseudo-template), vlastní streamy


wxDesigner - /home/robert/wxDesigner/code/wxDesigner.wdr
File Edit Dialog Element Menus Toolbars Bitmaps Help
wxDesigner.wdr
Dialogs
RegisterFunc
UnregisteredFunc
AddCppHandlerFunc
AddCppFileFunc
AddCppDialogFunc
SearchFunc
ReplaceFunc
AddPythonFileFunc
AddCppGetterFunc
wxBoxSizer V
wxBoxSizer H
wxFlexGridSizer
wxStaticText
wxChoice
wxStaticText
wxTextCtrl
wxStaticText
wxTextCtrl
wxStaticText
wxTextCtrl
wxStaticText
wxStaticText
wxStaticLine
wxBoxSizer V
wxBoxSizer H
wxStaticText
wxChoice
Foreign Control
wxStaticLine
wxStaticText
Dialog class:
Control type:
Control ID:
Method name:
Tip:
Choose a dialog on the right hand side and click on a control. The fields will get filled out automatically and a name for the method will be suggested from parsing the Id: ID_MY_TEXT will yield GetMyText() etc.
Dialog:
Add getter
Leave
Current selection: wxTextCtrl [ID name: ID_CON_CTRL, ID value: automatic, option: 0]


wxGlade v0.2
File View Help
wxGlade: Tree (/home/alb/pytho...
notebook_1_pane_1
sizer_3
use_menu_icons
frame_tool_win
use_dialog_units
show_progress
sizer_4
label_1
wxGlade: preferences
Interface
Use icons in menu items
Show properties and tree windows as small frames (Win32 only)
Use dialog units by default for size properties
Show progress dialog when loading wxg files
Initial path for file opening/saving dialogs:
Initial path for code generation file dialogs:
Number of items in file history (wxPython >= 2.3.3)
OK
Cancel
Apply
Properties - <frame_tool...
Common
Layout
Widget
Name
frame_tool_win
Class
wxCheckBox
Id
-1
Size
359, 23
Background
#d6d6d6
Foreground
#000000
Font
['12', 'default', 'norr
Tooltip

Hello World - GTK+

helloworld.c:

```
#include <gtk/gtkmain.h>
#include <gtk/gtkwindow.h>
#include <gtk/gtkbutton.h>
#include <glib/gi18n.h>
#include <glib/gprintf.h>

gchar *hellostr;

void window_destroy(GtkWidget *widget, gpointer data)
{
  gtk_main_quit ();
}

void button_clicked(GtkWidget *widget, gpointer data)
{
  g_printf ("%s\n", hellostr);
}

GtkWidget *helloworld_new()
{
  GtkWidget *window, *button;

  hellostr = g_locale_to_utf8 (_ ("Hello World"), -1, NULL, NULL, NULL);
```

```
  window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
  g_signal_connect (G_OBJECT (window), "destroy",
    G_CALLBACK (window_destroy), NULL);

  button = gtk_button_new_with_label (hellostr);
  g_signal_connect (G_OBJECT (button), "clicked",
    G_CALLBACK (button_clicked), NULL);
  g_signal_connect_swapped (G_OBJECT (button), "clicked",
    G_CALLBACK (window_destroy), G_OBJECT (window));
  gtk_container_add (GTK_CONTAINER (window), button);
  gtk_widget_show (button);

  return window;
}

int main(int argc, char *argv[])
{
  GtkWidget *helloworld;

  gtk_init (&argc, &argv);

  helloworld = helloworld_new();
  gtk_widget_show (helloworld);

  gtk_main ();

  return 0;
}
```

Hello World - gtkmm

main.cc:

```
#include "helloworld.h"
#include <gtkmm/main.h>

int main (int argc, char *argv[])
{
  Gtk::Main kit(argc, argv);

  HelloWorld helloworld;

  Gtk::Main::run(helloworld);

  return 0;
}
```

helloworld.h:

```
#ifndef HELLOWORLD_H
#define HELLOWORLD_H

#include <gtkmm/window.h>
#include <gtkmm/button.h>

class HelloWorld : public Gtk::Window
{
```

```
 public:
  HelloWorld();
  virtual ~HelloWorld();
 protected:
  Glib::ustring hellostr;
  Gtk::Button button;
  virtual void button_clicked();
};

#endif // HELLOWORLD_H
```

helloworld.cc:

```
#include "helloworld.h"
#include <glibmm/i18n.h>
#include <glib/gprintf.h>

HelloWorld::HelloWorld()
  : hellostr (Glib::locale_to_utf8 (_ ("Hello World"))),
    button(hellostr)
{
  button.signal_clicked().connect(sigc::mem_fun(*this,
&HelloWorld::button_clicked));
  button.signal_clicked().connect(sigc::mem_fun(*this,
&HelloWorld::hide));

  add(button);
  button.show();
}
```

```
HelloWorld::~HelloWorld()
{
}

void HelloWorld::button_clicked()
{
  g_printf ("%s\n", hellostr.c_str());
}
```

Hello World - Qt

main.cc:

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

int main (int argc, char *argv[])
{
  QApplication app(argc, argv);

  HelloWorld helloworld;

  app.setMainWidget (&helloworld);
  helloworld.show();
  app.exec();

  return 0;
}
```

helloworld.h:

```
#ifndef HELLOWORLD_H
#define HELLOWORLD_H

#include <qpushbutton.h>
#include <qmainwindow.h>
```

```
class HelloWorld : public QMainWindow
{
  Q_OBJECT
 public:
  HelloWorld();
  virtual ~HelloWorld();
 protected:
  QString hellostr;
  QPushButton button;
 protected slots:
  virtual void button_clicked();
};

#endif // HELLOWORLD_H
```

helloworld.cc:

```
#include "helloworld.h"
#include <qlayout.h>
#include <stdio.h>

HelloWorld::HelloWorld()
  : hellostr (QString::fromLocal8Bit (tr ("Hello World"))),
    button(hellostr, this)
{
  QObject::connect (&button, SIGNAL (clicked()), this,
    SLOT (button_clicked()));
  QObject::connect (&button, SIGNAL (clicked()), this, SLOT (close()));
```

```
  button.setSizePolicy (QSizePolicy::Minimum, QSizePolicy::Minimum);
  button.setMinimumSize (button.size());
  setCentralWidget (&button);
  adjustSize();
  button.show();
}

HelloWorld::~HelloWorld()
{
}

void HelloWorld::button_clicked()
{
  printf ("%s\n", hellostr.ascii());
}
```

Hello World - wxWidgets

main.cc:

```
#include "helloworld.h"
#include <wx/app.h>

class App: public wxApp
{
  virtual bool OnInit();
};

bool App::OnInit()
{
  HelloWorld *helloworld;

  helloworld = new HelloWorld();
  SetTopWindow(helloworld);
  helloworld->Show(TRUE);

  return TRUE;
}

IMPLEMENT_APP(App)
```

helloworld.h:

```
#ifndef HELLOWORLD_H
#define HELLOWORLD_H

#include <wx/frame.h>
#include <wx/button.h>

class HelloWorld : public wxFrame
{
 public:
  HelloWorld();
  virtual ~HelloWorld();
 protected:
  enum { ID_BUTTON = 1 };
  wxString hellostr;
  wxButton button;
  virtual void button_clicked(wxCommandEvent& WXUNUSED(event));
};

#endif // HELLOWORLD_H
```

helloworld.cc:

```
#include "helloworld.h"
#include <wx/intl.h>

HelloWorld::HelloWorld()
  : wxFrame(NULL, -1, _ ("")),
    hellostr (_ ("Hello World")),
    button (this, ID_BUTTON, hellostr)
```

```
{
  int w, h;

  Connect(ID_BUTTON, wxEVT_COMMAND_BUTTON_CLICKED,
          (wxObjectEventFunction)&HelloWorld::button_clicked);

  button.GetSize (&w, &h);
  button.Show();
  SetSizeHints (w, h);
  SetSize (w, h);
}

HelloWorld::~HelloWorld()
{
}

void HelloWorld::button_clicked(wxCommandEvent& WXUNUSED(event))
{
  printf ("%ls\n", hellostr.c_str());
  Close();
}
```

Architektura

Objekt

- bázová (většinou abstraktní) třída toolkitu, kořen **hierarchie tříd** (widgetů aj.)
- implementuje mechanizmus signálů a událostí a jejich obsluh/slotů, správu paměti objektů widgetů a další základní věci
- *GObject*, *Glib::ObjectBase*, *QObject*, *wxObject*
- v GTK+ není vícenásobná dědičnost, ale rozhraní, copy on write při kopírování objektu
- v gtkmm lze získat Gtk objekt pomocí *<widget>::gobj()*

Widget

- prvek GUI (tlačítka, nápisy, vstupní pole, scroll/tool/status/menubary, dialogy apod.)
- má stav, např. focus, default, sensitive (disabled)

- konstrukce GUI pomocí vnořování widgetů → **hierarchie widgetů** (kořenem hlavní (top-level) okno)
- stanovuje svoji minimální velikost, skutečnou velikost a polohu určuje dynamicky kontejner, ve kterém je obsažen (widget může mít nastavenu i pevnou velikost - size policy)
- *GtkWidget*, *Gtk::Widget*, *QWidget*, ve wxWidgets není společný předek (*wxObject*?)
- vytvoření (nezobrazí se):
  - v GTK+ *gtk_<widget>_new()*
  - v gtkmm *::kontruktor()*
  - v Qt *::konstruktor(rodic)*
  - ve wxWidgets *::konstruktor(rodic, . . .)*
- zobrazení:
  - v GTK+ *gtk_widget_show(w)* nebo *gtk_widget_show_all(w)*
  - v gtkmm *::show()* nebo *::show_all()*
  - v Qt *::show()*

  - ve wxWidgets *::Show()*

- schování:
  - v GTK+ *gtk_widget_hide(w)*
  - v gtkmm a Qt *::hide()*
  - ve wxWidgets *::Hide()*

- zrušení/zavření:
  - v GTK+ *gtk_widget_destroy(w)*
  - v gtkmm *::hide()* a zrušení ručně nebo kontejnerem
  - v Qt *::close()*
  - ve wxWidgets *::Close()*

Okno

- top-level widget (hlavní okno, dialog)
- využívá jedno nebo více X oken (zdroj X serveru)
- *GtkWindow*, *Gtk::Window*, *QMainWindow* (*QWidget*), *wxFrame*

Kostra aplikace

- na začátku programu inicializace toolkitu (případně odebrání specifických parametrů příkazové řádky):
  - v GTK+ *gtk_init(&argc, &argv)*
  - v gtkmm vytvoření jediného objektu třídy *Gtk::Main (argc, argv)*
  - v Qt vytvoření jediného objektu třídy *QApplication (argc, argv)*
  - wxWidgets se neinicializuje
- "poskládání" widgetů (do oken, pomocí kontejnerů) a "napojení" signálů a událostí na osluhy/sloty
- spuštění hlavní smyčky:
  - v GTK+ *gtk_main()*
  - v gtkmm *Gtk::Main::run(<hlavní_widget>)*
  - v Qt *QApplication:exec()* (po nastavení hlavního widgetu pomocí *QApplication::setMainWidget(w)*)
  - ve wxWidgets přetížení metody *wxApp::OnInit()* a v ní vytvoření hlavního widgetu, makro *IMPLEMENT_APP(<třída_zděděná_z_wxApp>)*

- ukončení hlavní smyčky (většinou následuje ukončení programu):
  - v GTK+ *gtk_main_quit()*
  - v gtkmm, Qt a wxWidgets zrušení/zavření hlavního widgetu

Signály a události, obsluhy signálů a událostí (signal and event handlers, v GTK+ a wxWidgets)/sloty (v gtkmm a Qt)

- widget vysílá **signál**, když se s ním něco stane (např. kliknutí na tlačítko), také ruční vyslání signálu v aplikaci, zpracovávány synchronně (ihned se vykoná obsluha)
- X server posílá **události** o aktivitě uživatele (např. stisk klávesy nebo pohyb myši), X serveru (např. žádost o překreslení okna) nebo WM (např. změna velikosti okna), také ruční vyvolání události aplikací (např. překreslení), zpracovávány asynchronně (fronta, "vyzvednutí" v hlavní smyčce a vykonání obsluhy), mohou se propagovat do rodičovského widgetu
- signály a události jsou zpracovány **obsluhami/sloty** (reakce aplikace na akci) - v GTK+ funkce, v gtkmm, Qt a wxWidgets funkce nebo metoda třídy, existují výchozí metody, "zabalení" funkce nebo metody do podoby obsluhy/slotu:

  - v GTK+ *G_CALLBACK(func)*
  - v gtkmm *sigc::mem_fun(&metoda)* nebo *sigc::ptr_fun(&func)* jako funktor (přetížený *operator()*)
  - v Qt *SLOT(metoda)*
  - ve wxWidgets *(wxObjectEventFunction)&metoda*

- v GTK+ a gtkmm se události interně překládají na signály, obsluhy/sloty se vykonávají v pořadí registrace

- v Qt se mohou události zpracovávat i synchronně, sloty se vykonávají v nedefinovaném pořadí, filtry na události - objekt jako filtr událostí pro jiný objekt

- ve wxWidgets jsou jen události a mohou mít jen 1 obsluhu (!)

- registrace/připojení obsluhy/slotu pro konkrétní objekt (obvykle widget):
  - v GTK+ *g_signal_connect(G_OBJECT(o), sig/ev, G_CALLBACK(f), data)*
  - v gtkmm *::signal_<signál/událost>()::connect(sigc::mem/ptr_fun(&f/m))* nebo přetížení metody *::on_<signál>/<událost>()*

  - v Qt *QObject::connect(&o, SIGNAL(sig/ev), &obj_příjemce, SLOT(m))*
  - ve wxWidgets (dynamicky) *::Connect(id_obj, ev, (wxObjectEventFunction)&m)*, nebo (staticky) makra *BEGIN_EVENT_TABLE(<třída>, <třída_widgetů>)*, *EVT_MENU(ev, m)*, *EVT_BUTTON(ev, m)*, ... a *END_EVENT_TABLE()*

- "ruční" vyslání signálu nebo události:
  - v GTK+ *g_signal_emit()*
  - v gtkmm *::signal_<signál/událost>()*
  - v Qt emit *::<metoda_signálu/události>()* (emit lze vynechat), *::sendEvent()*, *::postEvent()*
  - ve wxWidgets nelze (?)

Správa paměti

- kontejner nebo rodič může spravovat obsažené/podřízené objekty → **hierarchie objektů** - běžně kopíruje hierarchii widgetů
- v GTK+ počítání referencí na objekt, při vytvoření 1, zrušen při 0, *g_object_ref(o)*, *g_object_unref(o)*, kontejnery ruší obsažené widgety

- v gtkmm počítání referencí na objekt pomocí tzv. smartpointeru *Glib::RefPtr*, rušení buď ručně nebo kontejnerem (vložení ne přímo objektu widgetu, ale *manage(objekt)*)

- v Qt a wxWidgets při vzniku objektu zadaván rodič (většinou kontejner), který ho ruší

Konstrukce GUI

- v GTK+ úpravy/skládání widgetů do kontejnerů a navazování obsluh signálů ve funkcích vytvářejících widget

- v gtkmm, Qt a wxWidgets úpravy/skládání widgetů do kontejnerů v konstruktoru nové třídy zděděné z třídy widgetů (nového widgetu), (přetížené) metody pro obsluhy/sloty = **subclassing**

Widgety a ostatní prvky (G)UI

Kontejner

- widget, který umožňuje vnořit jiné widgety → hierarchie widgetů
- tvorba GUI pomocí skládání widgetů do kontejnerů
- může (ale nemusí) rušit/zavírat vnořené widgety při svém zrušení/zavření, včetně dealokace paměti
- dynamicky určuje skutečnou velikost a polohu vnořených widgetů na základě jejich požadavků (minimální) velikosti
- ve wxWidgets jsou na určování skutečné velikosti a polohy vnořených widgetů **sizery** → separátní hierarchie
- většina widgetů (prvků GUI) jsou kontejnery, speciální vybrané:
  - v GTK+ *GtkContainer*, *GtkBin*, *Gtk*Box*, *GtkTable*, *GtkFixed*, *GtkLayout*
  - v gtkmm stejné jako v GTK+, jen místo prefixu *Gtk* namespace *Gtk::*

  - v Qt *Q*Box*, *QTable*, *QGrid* a *QLayout* (není widget, podobné sizerům ve wxWidgets)
  - ve wxWidgets sizery *wxSizer*, *wx*BoxSizer*, *wx*GridSizer*

- vládání/odebírání widgetů:
  - v GTK+ speciální funkce pro každý kontejner, *gtk_container_add(c, w)*, *gtk_container_remove(c, w)*
  - v gtkmm speciální funkce pro každý kontejner, *Gtk::Container::add(w)*, *Gtk::Container::remove(w)*
  - v Qt uvést kontejner jako rodiče při vytváření widgetu, *::reparent(p, . . .)*, *QLayout::add(w)*, *QLayout::remove(w)*
  - ve wxWidgets uvést kontejner jako rodiče při vytváření widgetu, *::Reparent(p)*, *wxSizer::Add(w, . . .)*, *wxSizer::Remove(w)*

Prvky GUI (základní)

- vybrané, pro kompletní přehled viz referenční manuály

- v GTK+ *GtkButton* (*GtkToggleButton*, *GtkCheckButton*, *GtkRadioButton*), *GtkLabel*, *GtkEntry*, *GtkSpinButton*, *GtkCombo*, *GtkOptionMenu*, *GtkImage*, *GtkFrame*, *GtkProgressBar*, *GtkMenuBar*, *GtkMenu*, *GtkMenuItem*, *GtkToolbar*, *GtkStatusbar*, *GtkScrollbar*, *GtkHandleBox*, *GtkTooltips*, *GtkTreeView*, *GtkTextView*, *GtkNotebook*, *GtkScrolledWindow*, *GtkWindow*, *GtkDialog*, *GtkFileSelection* (. . .), *GtkMessageDialog*

- v gtkmm stejné jako v GTK+, jen místo prefixu *Gtk* namespace *Gtk::*

- v Qt *QPushButton*, *QCheckBox*, *QRadioButton*, *QLabel*, *QLineEdit*, *QSlider*, *QSpinButton*, *QComboBox*, *QListBox*, *QPixmap*, *QImage*, *QGroupBox*, *QProgressBar*, *QMenuBar*, *QPopupMenu*, *QToolBar*, *QStatusLine*, *QScrollBar*, *QToolTip*, *QListView*, *QTextEdit*, *QTabWidget*, *QScrollView*, *QMainWindow*, *QDialog*, *QFileDialog* (. . .), *QMessageBox*

- ve wxWidgets *wxButton* (*wxToggleButton*, *wxCheckBox*, *wxRadioButton*), *wxStaticText*, *wxStaticLine*, *wxSlider*, *wxSpinButton*, *wxComboBox*, *wxChoice*, *wxBitmap*, *wxImage*, *wxGauge*, *wxMenuBar*, *wxMenu*, *wxToolBar*, *wxStatusBar*, *wxScrollBar*, *wxToolTip*, *wxListCtrl*, *wxTreeCtrl*, *wxTextCtrl*, *wxNotebook*, *wxScrolledWindow*, *wxFrame*, *wxWindow*, *wxDialog*, *wxFileDialog* (. . .), *wxMessageDialog*

Menu

- hlavní i popup, akcelerátory (A-písmeno)
- kromě skládání widgetů i generování menu ze struktury

Prvky bez X okna

- kreslí do okna rodiče
- nedostávají události (lze obejít pomocí widgetů **Event**)

Ostatní (pokročilé) prvky

- různé toolkity poskytují různé další prvky
- časovače (timeouty), idle funkce, asynchronní I/O
- textové (konfigurační) soubory pro nastavení vzhledu/stylu (fonty, barvy, pozadí), dynamické změny za běhu - v GTK+, gtkmm a wxWidgets resource soubory, v Qt *QSettings*

Kreslení

- widgety pro jednoduchou oblast pro kreslení (čar, obdélníků, oblouků apod.), umisťování grafických prvků na **canvas** a zobrazení výstupu **OpenGL**
- v GTK+ widgety *GtkDrawingArea*, knihovny *Cairo*, *GnomeCanvas* (zastaralá), *GooCanvas*, *clutter* (OpenGL)
- v gtkmm widgety *Gtk::DrawingArea*, knihovny *GnomeCanvas* (zastaralá), *GooCanvas*, *clutter* (OpenGL)
- v Qt widgety *QPainter*, *QGraphicsScene* (framework Graphics View) a *QGLWidget* (z modulu QtOpenGL)
- ve wxWidgets *wxPaintDC* (lze kreslit na libovolný widget), knihovna *Open Graphics Library* (*OGL*), *wxGLCanvas*

Komunikace mezi aplikacemi

- díky architektuře X Window Sytem i síťově
- selekce - např. blok textu, widgety *GtkClipboard*, *Gtk::Clipboard*, *QClipboard*, *wxClipboard*
- drag&drop - widgety pro zdroj a cíl, složitější architektura

Vytvoření nového widgetu (třídy widgetů)

- v GTK+ poměrně složitě pomocí subclassingu v C - definice 2 struktur (třídy a instance), registrace typu, funkce pro inicializaci, rušení, kreslení atd. objektu $\rightarrow$ v praxi se nepoužívá (jen ve vlastní knihovně GTK+)
- v gtkmm, Qt a wxWidgets jednoduše děděním z třídy widgetů (subclassing), v konstruktoru nové třídy se widget "poskládá" $\rightarrow$ používá se pro konstrukci GUI
- v Qt v deklaraci nové třídy makro *Q_OBJECT* v části private:, další části *public/protected/private slots:* a *signals:* (kvůli MOS), pre-překlad do C++ pomocí `moc` (do souboru `moc_*.cpp`)
- ve wxWidgets v deklaraci nové třídy makro *DECLARE_EVENT_TABLE()* v části private:, pokud je použita statická registrace/připojení obsluhy událostí (makra *BEGIN_EVENT_TABLE*, *EVT_** a *END_EVENT_TABLE*)

Internacionalizace (i18n) a lokalizace (l10n)

- toolkity interně pracují s texty a znaky v kódování UTF8
- v GTK+ a gtkmm standardně pomocí **GNU Gettext** (*setlocale()*, *bindtextdomain()*, *textdomain()*, *_("")*, `xgettext` → `*.pot`, `msgmerge` → `*.po`, `msgfmt` → `*.mo`), konverze textů do UTF8 pomocí *g_locale_to_utf8(_(""))*, *Glib::locale_to_utf8(_(""))*
- v Qt vlastní řešení (*QTranslator::load()*, *QApplication::installTranslator()*, *tr("")*, `lupdate` → `*.ts`, Qt Linguist, `lrelease` → `*.qm`) nebo pomocí GNU Gettext a konverze textů do UTF8 pomocí *QString::fromLocal8Bit(_(""))*
- ve wxWidgets pomocí *wxLocale* (*wxLocale::Init()*, *wxLocale::AddCatalog()*) a GNU Gettext (předefinované *_("")* včetně konverzí do UTF8)

Hello World

Už jsou zdrojové kódy ukázkové aplikace Hello World jasnější?

Knihovny

GTK+

- *GTK* - widgety a ostatní prvky (G)UI (*gtk_**)
- *GDK* - wrapper nad Xlib (*gdk_**), nízkoúrovňová grafika
- *GObject* - typový, objektový a signálový systém (*GObject*)
- *GLib* - základní typy (*gint32*, *gpointer*, atd.), datové struktury (*g_str*()*, *GString*, *GList*, *GTree*), správa paměti (*g_malloc()*, *g_new()*, *g_free()*), I/O (*GDir*), vlákna (*GThread*, *GMutex*), I/O kanály (*GIOChannel*), i18n a l10n
- *theme engines* - grafická podoba widgetů, změny za běhu
- *Pango*, *Cairo* - renderování textu a 2D vektorové grafiky
- *ATK* - přístupnost
- a další

gtkmm

- *gtkmm* - widgety a ostatní prvky (G)UI (*Gtk::*)
- *gdkmm* - wrapper nad Xlib (*Gdk::*), nízkoúrovňová grafika
- *libsigc++* - knihovna implementující signálový systém (*sigc::*), http://libsigc.sf.net
- *glibmm* - typový a objektový systém (*Glib::Object*), základní typy (*Glib::Value**), datové struktury (*Glib::ustring*), I/O (*Glib::Dir*), vlákna (*Glib::Thread*, *Glib::Mutex*), i18n a l10n
- *Pango::*, *Atk::* a další

Qt

- vše v jedné knihovně (SDK, Software Development Kit), moduly QtCore, QtGui aj.
- widgety a ostatní prvky (G)UI, i18n a l10n
- grafika a multimedia - 2D i 3D (OpenGL), Phonon framework
- šablony datových struktur (*Qt Template Library*) - unicode řetězce (*QString*) seznamy, zásobník, vektor, mapa, . . .

- XML - parsování pomocí SAX2 (*QXML**) i DOM2 (*QDom**)
- web - HTML, CSS, Javascript (*QWebKit*)
- I/O (*QFile*, *QDir*)
- sítě (*QSocket*, *QDns*, *QHttp*)
- vlákna (*QProcess*, *QThread*, *QMutex*, *QSemaphore*)
- databáze (*QtSql*)
- skriptivání (*QtScript*)
- a další

wxWidgets

- vše v knihovnách *wxBase* a systémové (porty *wxGTK*, *wxMSW*, atd.)
- widgety a ostatní prvky (G)UI, i18n a l10n
- unicode řetězce (*wxString*), datové struktury (*wxList*, *wxHashMap*)
- HTML (*wxHtmlParser*)

- I/O (*wxFile*, *wxDir*, *wxDllLoader*)
- vlákna (*wxProcess*, *wxThread*, *wxMutex*, *wxSemaphore*)
- sítě (*wxIPV4address*, *wxHTTP*)
- databáze - ODBC
- tvorba nápovědy - M$ HTML a Windows Help, wxWidgets HTML Help, aplikace HelpBlocks, widget *wxHelpController*

Překlad aplikace

- nástroj pro grafický návrh GUI ("designer", Glade(mm), Qt Designer, wx-Designer), který je případně součástí IDE (**Anjuta**, **Qt Creator**)
- "ručně" překladačem (+ make) s pomocí nástrojů pro cesty k hlavičkovým souborům a knihovnám

GTK+ (C)

`#include <gtk/gtk.h>` nebo pro jednotlivé widgety (např. `gtk/gtkbutton.h`)

```
pkg-config --cflags --libs gtk+-2.0
```

Překladače: GCC, MinGW (případně Cygwin), M$VC 6 (použita knihovna MSVCRT)

gtkmm (C++)

`#include <gtkmm.h>` nebo pro jednotlivé widgety (např. `gtkmm/button.h`)

```
pkg-config --cflags --libs gtkmm-2.4
```

Překladače: GCC, MinGW, M$VC++ $\geq$ 2005

Qt (C++)

`#include <<widget>.h>` pro jednotlivé widgety (např. `qpushbutton.h`)

`qmake` a `make`

- generuje `Makefile` ze souboru projektu (popisuje, které zdrojové soubory jsou potřeba pro překlad programu), který se vytvoří pomocí `qmake -project`
- potřeba nastavit proměnné prostředí `QTDIR` (adresář s Qt) a `QMAKESPEC` (kombinace platformy a kompilátoru, např. linux-g++, win32-g++ nebo macx-g++)

"ručně":

```
moc -o moc_*.cc *.h
```

`-I<qt_include_dir> -L<qt_lib_dir> -lqt` (nebo `-lqt-mt`)

Překladače: GCC, MinGW, M$VC++ $\geq$ 2003

wxWidgets (C++)

`#include <wx/wx.h>` nebo pro jednotlivé widgety (např. `wx/button.h`)

```
wx-config --cxxflags --libs
```

Překladače: GCC, MinGW (potřeba MSYS), Cygwin, M$VC++ $\geq$ 6, CodeWarrior, Apple Developer Tools (založeny na GCC)