Glade на новый

ЧАСТЬ 5 Описать интерфейс программы в XML и подгрузить его на лету? Совершенно необязательно покупать Visual Studio и устанавливать .NET 3.0 – **Андрей Боровский** покажет, как решить задачу средствами Glade!

Компьютеры бесполезны, ибо они дают только ответы.

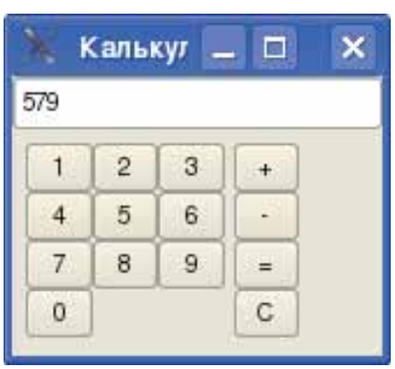
Сальвадор Дали

Если бы великий художник лучше разбирался в компьютерах, он сумел бы найти в них гораздо больше недостатков. Чем занимаются большую часть времени настольные компьютеры? Они выполняют работу более примитивных устройств. Компьютер может быть видеомагнитофоном, радиоприемником, будильником и, конечно, конечно, пишущей машинкой. Ну а кроме всего прочего, любая операционная система на любой платформе предлагает программу, выполняющую функции простейшего микрокалькулятора. Непреложный факт – микрокалькулятор обязательно будет сэмулирован на любой системе, достаточно мощной доля того, чтобы его эмулировать. Нет, я, конечно, понимаю, что компьютерные проигрыватели, компьютерные будильники и компьютерные микрокалькуляторы полезны. Я и сам с удовольствием ими пользуюсь. Я лишь хочу сказать, что компьютеры все еще по-настоящему не революционизировали нашу жизнь. Ну а в ожидании революции, после которой все будет не так, как было, мы, уважаемый читатель, тоже напишем простейшую программу-калькулятор, и воспользуемся для этого, естественно, инструментарием *GTK+*.

В предыдущей статье мы научились создавать проекты графических приложений с помощью *Glade*. Используя *Glade 2.x*, мы сгенерировали не только описание интерфейса программы, но и заготовку ее исходного кода, включая вспомогательные функции и обработчики событий. Сегодня мы рассмотрим другой способ работы с *Glade*. При новом подходе *Glade* используется исключительно для проектирования интерфейса, а исходный код приложения пишется программистом с помощью других средств (по-видимому, этот способ работы станет единственно возможным в *Glade 3.x*).

Описание интерфейса, созданное *Glade* в формате XML, и исходный код программы связываются между собой с помощью функций библиотеки *libglade*. Все это немного похоже на систему XMLGUI, реализованную в KDE. Разница заключается в том, что в XMLGUI внешний XML-документ описывает лишь отдельные элементы интерфейса программы, тогда как при работе с *libglade* все описание интерфейса загружается из XML-файла [Это не означает, что в Qt/KDE нет технологии, аналогичной libglade. Интерфейсы, созданные в Qt Designer, также можно загружать и обрабатывать на лету, – прим. ред.].

(Рис. 1) Программа-микрокалькулятор.



Приступим к проектированию

Для того, чтобы понять, как работает библиотека *libglade*, необходимо сначала разобраться, что представляет собой проект Glade. Разбираться, как вы уже поняли, мы будем на примере программы-микрокалькулятора (**Рис. 1**).

Как можно заметить, наш калькулятор довольно примитивен. Поддерживаются только операции сложения и вычитания, причем над целыми числами. Отсутствует контроль переполнения при выполнении арифметических действий. При желании вы можете нарастить функциональность нашего микрокалькулятора сами – добавить поддержку чисел с плавающей точкой, тригонометрические функции, встроенный язык программирования... Нет пределов совершенству. Мы же сосредоточимся на программировании интерфейса калькулятора средствами *GTK+*, *Glade* и *libglade*.

Как это обычно бывает при работе со средствами визуального программирования, проектирование программы начинается с пользовательского интерфейса. В главном окне приложения мы размещаем несколько контейнеров для более удобной упаковки элементов калькулятора – индикаторной панели и кнопок.

Свойству «Имя» объекта главного окна мы присваиваем значение rootwnd. Это имя нам нужно запомнить, так как оно будет играть важную роль при взаимодействии с библиотекой *libglade*. В верхней части главного окна расположено текстовое поле GtkEntry, которое будет служить индикаторной панелью калькулятора. Кнопки калькулятора – это объекты GtkButton. Вот, собственно, и все, других элементов управления наш виртуальный калькулятор (как и его «железный» собрат) не предполагает.

Иерархию объектов-контейнеров и элементов управления, из которых состоит калькулятор, проще (и полезнее) показать, нежели описать. Для этого нужно всего лишь открыть окно «Дерево эл. управления» среды *Glade* (**Рис. 2**). Перед вами появится древовидный список всех визуальных элементов.

Объекты-контейнеры являются внутренними узлами дерева, а элементы управления – его «листьями». Если мы теперь сохраним проект *Glade* под именем calculator, на диске появится файл **calculator.glade**. Именно этот файл содержит описание спроектированного нами графического интерфейса программы. Если мы откроем файл **calculator.glade** в текстовом редакторе, то увидим примерно следующее:

```
<?xml version="1.0" standalone="no"?> <!--*- mode: xml -*-->
<!DOCTYPE glade-interface SYSTEM "http://glade.gnome.org/glade-2.0.dtd">

<glade-interface>

<widget class="GtkWindow" id="rootwnd">
  <property name="visible">True</property>
  <property name="title" translatable="yes">Калькулятор</property>
```

Месяц назад Мы изучали визуальное программирование в духе старой школы *Glade 2.x*.


```
...
  <signal name="destroy" handler="gtk_main_quit" last_modification_
 time="Tue, 13 Mar 2007 21:18:21 GMT"/>

  <child>
    <widget class="GtkVBox" id="vbox1">
      <property name="visible">True</property>
      ...
      <child>
              <widget class="GtkEntry" id="entry1">
                <property name="visible">True</property>
                ...
      </child>
...
```

Таким образом, файл ***.glade** представляет собой XML-документ, в котором содержатся списки значений свойств каждого объекта интерфейса, а иерархия интерфейса реализована при помощи иерархии вложенных тегов.

Немного кода

Теперь, когда у нас есть описание графического интерфейса, созданное *Glade*, нам будет очень просто написать демонстрирующую его программу. Открою небольшой секрет – функциональность *libglade* используется самой средой *Glade*, которая должна отображать создаваемый интерфейс в отдельном окне. Исходный код программы, загружающей и отображающей графический интерфейс, описанный в файле calculator.glade, состоит примерно из десяти строк (вы найдете его на диске в файле **calculator-1.c**).

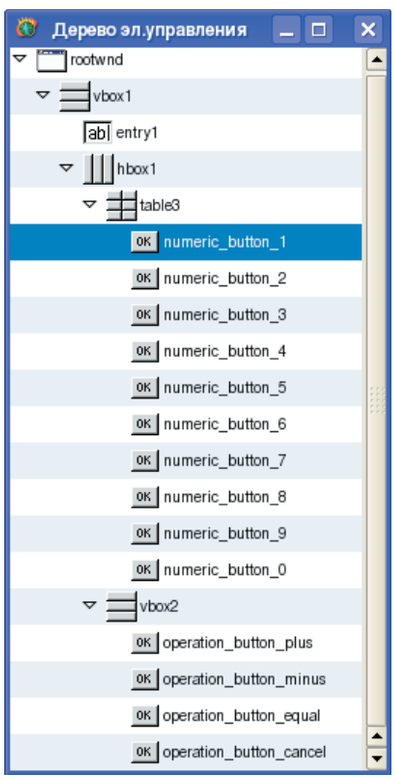
```
#include <stdlib.h>
#include <stdio.h>
#include <gtk/gtk.h>
#include <glade/glade.h>

int main (int argc, char **argv)
{
  GladeXML *xml;
   gtk_init(&argc, &argv);
   glade_init();
   xml = glade_xml_new("calculator.glade", "rootwnd", NULL);
   if (!xml) {
            g_warning("Failed to create the interface");
            return 1;
   }
   gtk_main();
   return 0;
}
```

Обратите внимание, что теперь, помимо заголовочного файла **gtk/gtk.h** мы включаем в программу файл **glade/glade.h**. Он содержит объявления функций и типов данных библиотеки *libglade*, а именно она будет служить рабочей лошадкой нашей программы. Собственно программа начинается с вызова знакомой нам функции gtk_init(). Далее мы вызываем новую функцию glade_init(). Ее задача заключается в том, чтобы инициализировать систему *libglade*. Связывание программы с XML-файлом, описывающим ее интерфейс, выполняется функцией glade_xml_new(). Первым аргументом этой функции должен быть файл, созданный в *Glade*, вторым – имя корневого элемента иерархии графических объектов интерфейса, определенного в этом файле. Третий параметр glade_xml_new() описывает домен перевода приложения (translation domain). В качестве значения этого параметра мы можем передать NULL. Функция glade_xml_new() возвращает указатель на структуру *GladeXML*, который мы сохраняем в переменной xml. Структура *GladeXML* инкапсулирует описание интерфейса, созданного *Glade*. Библиотека *libglade* экспортирует несколько функций, имена которых начинаются с префикса glade_xml_. Эти функции позволяют управлять элементами интерфейса, созданного с помощью *Glade*, и каждой из них в качестве одного из параметров следует передавать указатель на структуру *GladeXML*.

Впрочем, в версии **calculator-1.c** значение переменной **xml** еще не востребовано. Вызов glade_xml_new() приводит к загрузке файла описания интерфейса в программу, а библиотека *libglade* позаботится о его правильном отображении. Нам остается только запустить цикл обработки сообщений с помощью функции gtk_main(). Скомпилируем нашу программу, используя следующую команду:

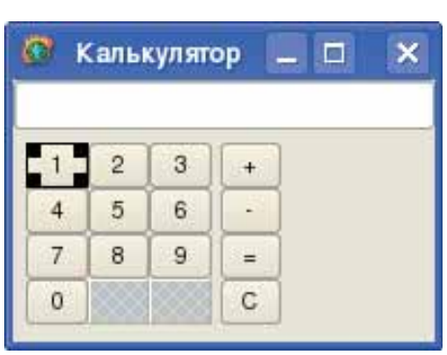
```
gcc calculator-1.c -o calculator `pkg-config --cflags --libs libglade-2.0`
```

На этот раз в качестве аргумента утилиты *pkg-config* мы используем имя пакета *libglade-2.0*. Теперь можно запустить программу *calculator*. Вы, конечно, сразу заметите, что хотя программа правильно отображает все элементы пользовательского интерфейса, она ничего не делает (даже завершиться как следует не умеет). Это вполне естественно, ведь мы еще не определили ни одного обработчика сигнала.



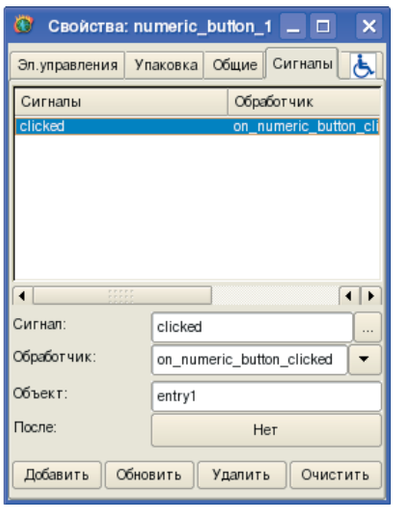
(Рис. 2) Окно просмотра структуры элементов интерфейса.

Снова откройте (если, вдруг, вы его закрыли) проект *calculator* в среде *Glade*. Прежде всего, выделим в редакторе свойств объект rootwnd, перейдем на вкладку «Сигналы» редактора свойств и создадим обработчик сигнала destroy. В качестве процедуры обработчика из раскрывающегося списка «Обработчик» выберем функцию gtk_main_quit(). Вот так просто можно связать обработчик сигнала закрытия окна и функцию завершения работы программы. Далее выделим в редакторе форм одну из цифровых кнопок калькулятора (**Рис. 3**).



(Рис. 3) Окно редактора формы с выделенной кнопкой.

Всем цифровым кнопкам присвоены имена вида numeric_button_x,



(Рис. 4) Окно редактора свойств в режиме создания обработчика сигнала.

где x – число от 0 до 9. Перейдем на вкладку «Сигналы» редактора свойств и создадим для выбранной кнопки обработчик сигнала clicked. Процедура обработчика получит имя on_numeric_button_x_clicked(). Переименуем обработчик в on_numeric_button_clicked(). При назначении обработчика каждой кнопке в поле ввода «Объект» укажем значение entry1 (**Рис. 4**).

Напомню, что поле ввода «Объект» позволяет задать имя объекта, указатель на который будет передан процедуре обработчика в качестве дополнительного параметра. Объект entry1 – это объект класса GtkEntry – «индикаторная панель» нашего калькулятора. Вполне естественно, что обработчики событий clicked цифровых кнопок должны иметь доступ к объекту, представляющему индикаторную панель. Мы переименовали обработчик сигнала clicked в on_numeric_button_clicked(), чтобы подчеркнуть, что у нас будет один обработчик сигнала clicked для всех цифровых кнопок.

Назначим теперь обработчик on_numeric_button_clicked() сигналам clicked всех кнопок с цифрами (не забудем указать объект entry1 в качестве дополнительного параметра обработчика каждой кнопки). Кнопки +, –, = и C имеют имена operation_button_plus, operation_button_minus, operation_button_equal и operation_button_cancel соответственно. В среде *Glade* создадим для каждой кнопки свой обработчик события clicked (соответственно функции on_operation_button_plus_clicked(), on_operation_button_minus_clicked(), on_operation_button_equal_clicked() и on_operation_button_cancel_clicked()). Обработчикам on_operation_button_equal_clicked() и on_operation_button_cancel_clicked() в качестве дополнительного параметра должен быть передан объект entry1. Сохраним проект *Glade*. На этом визуальное программирование калькулятора закончено, и нам остается довершить воплощение наших идей в коде. Вариант программы с обработчиками сигналов вы найдете в файле **calculator-2.c**, основанном на файле **calculator-1.c**.

Доведем до совершенства!

Прежде всего нам нужно написать функции-обработчики сигналов:

```
void on_numeric_button_clicked(gpointer user_data, GtkButton *button)
{
  int i = atoi(gtk_button_get_label(button));
  if (mode == COPY_MODE) {
    bgvalue = fgvalue;
    fgvalue = 0;
    mode = INPUT_MODE;
  }
  fgvalue = fgvalue*10 + i;
  sprintf(screen, "%li\0", fgvalue);
  gtk_entry_set_text(GTK_ENTRY(user_data), screen);
}

void on_operation_button_plus_clicked(GtkButton *button)
{
  mode = COPY_MODE;
  op_state = OP_PLUS;
}

void on_operation_button_minus_clicked(GtkButton *button)
{
  mode = COPY_MODE;
  op_state = OP_MINUS;
}

void on_operation_button_equal_clicked(gpointer user_data, GtkButton
*button)
{
  switch (op_state) {
    case OP_PLUS:
      fgvalue += bgvalue;
      break;
    case OP_MINUS:
      fgvalue = bgvalue - fgvalue;
    default:;
  }
  sprintf(screen, "%li\0", fgvalue);
  gtk_entry_set_text(GTK_ENTRY(user_data), screen);
  mode = COPY_MODE;
}

void on_operation_button_cancel_clicked(gpointer user_data, GtkButton
*button)
{
  bgvalue = fgvalue = 0;
  mode = COPY_MODE;
  op_state = OP_NONE;
  sprintf(screen, "%li\0", fgvalue);
  gtk_entry_set_text(GTK_ENTRY(user_data), screen);
}
```

Необходимо указать одну особенность взаимодействия программы *libglade* с обработчиками сигналов. Если при связывании сигнала с обработчиком указывается дополнительный параметр, этот параметр будет первым аргументом функции-обработчика. Вторым аргументом будет указатель на объект-источник сигнала. Например, в обработчике on_numeric_button_clicked() первый аргумент указывает на объект entry1, а второй аргумент – на объект numeric_button_x, для которого вызван обработчик. Если же при связывании обработчика с сигналом дополнительные параметры не указываются, первым (и единственным) аргументом функции-обработчика будет указатель на объект-источник сигнала. Например, в обработчике on_operation_button_plus_clicked() первым аргументом является указатель на объект operation_button_plus, для которого вызван обработчик.

Вдаваться в подробности работы калькулятора мы не будем. Во-первых, они очевидны. Во-вторых, наша статья посвящена не написанию калькуляторов, а использованию *libglade*, так что мы рассмотрим лишь некоторые аспекты взаимодействия обработчиков с элементами интерфейса.

В обработчике on_numeric_button_clicked() нам необходимо получить цифру, соответствующую нажатой кнопке. Мы не можем передавать цифру в дополнительном параметре обработчика, поскольку этот параметр уже занят объектом entry1. Вместо этого мы просто считываем цифру, являющуюся меткой нажатой кнопки, с помощью функции gtk_button_get_label(). Текст элемента ввода entry1 устанавливается с помощью функции gtk_entry_set_text().

Хотя мы и создали обработчики всех сигналов, но если бы мы скомпилировали программу на данном этапе, ее элементы управления все равно ничего бы не делали. Для того чтобы наладить в программе обработку сигналов, недостаточно написать их обработчики. Необходимо решить еще одну проблему, которая в явном или неявном виде возникает во всех средах визуального программирования. В описании интерфейса программы мы указали, например, что обработчиком события clicked для кнопки operation_button_minus является нечто по имени on_operation_button_minus_clicked(). На этапе визуального программирования строка on_operation_button_minus_clicked() не является именем функции, поскольку никакой функции еще нет. Фактически мы просто добавили в описание интерфейса программы имя обработчика сигнала. Если бы связывание сигнала и обработчика выполнялось на этапе компоновки программы (как это происходит в «настоящих» IDE), компоновщик нашел бы функцию on_operation_button_minus_clicked() и связал бы ее с сигналом. Но в *GTK+* связывание сигналов и обработчиков происходит во время выполнения программы. Каким же образом программа находит функцию, соответствующую имени обработчика?

Ведь на этапе выполнения программы система *libglade* не знает имен функций (а знает только их адреса) и не может сопоставить функцию с именем обработчика. Для решения этой проблемы мы должны явным образом указать, что функция on_operation_button_minus_clicked(), определенная нами в файле **calculator-2.c**, является «тем самым» обработчиком on_operation_button_minus_clicked(). Мы делаем это с помощью функции glade_xml_signal_connect(). Первым параметром glade_xml_signal_connect() должен быть указатель на структуру *GladeXML*. Вторым параметром функции является имя обработчика, заданное при описании интерфейса в среде *Glade*, а третьим параметром – адрес определенной нами функции-обработчика. Из сказанного следует, между прочим, что функция-обработчик сигнала не обязана иметь то же имя, что и обработчик, назначенный сигналу в файле описания интерфейса. Важно лишь установить соответствие между двумя именами (точнее, между именем обработчика и адресом функции). Таким образом, в функцию main() файла **calculator-2.c** нужно добавить следующие строки, в которых устанавливается соответствие между именами обработчиков, заданными в файле **calculator.glade**, и функциями-обработчиками, определенными в программе:

```
glade_xml_signal_connect(xml, “gtk_main_quit”, G_CALLBACK(gtk_main_quit));
  glade_xml_signal_connect(xml, “on_numeric_button_clicked”, G_CALLBACK(on_numeric_button_clicked));
  glade_xml_signal_connect(xml, “on_operation_button_plus_clicked”, G_CALLBACK(on_operation_button_plus_clicked));
  glade_xml_signal_connect(xml, “on_operation_button_minus_clicked”, G_CALLBACK(on_operation_button_minus_clicked));
  glade_xml_signal_connect(xml, “on_operation_button_equal_clicked”, G_CALLBACK(on_operation_button_equal_clicked));
  glade_xml_signal_connect(xml, “on_operation_button_cancel_clicked”, G_CALLBACK(on_operation_button_cancel_clicked));
```

Теперь вы можете скомпилировать программу и убедиться, что все кнопки (включая кнопку закрытия окна, завершающую приложение) работают как надо.

Помимо функции glade_xml_signal_connect(), библиотека *libglade* экспортирует функцию glade_xml_signal_connect_data(). От glade_xml_signal_connect() эта функция отличается тем, что позволяет указать объект, который будет передаваться обработчику сигнала в качестве дополнительного параметра. В нашей программе мы определили передачу объекта entry1 обработчикам сигналов еще на этапе визуального программирования. Однако может случиться так, что объект, который следует передать обработчику сигнала, станет известен только во время выполнения программы. В этой ситуации функция glade_xml_signal_connect_data() окажется более полезной.

API libglade предоставляет нам еще несколько функций, связанных с обработкой сигналов. Функция glade_xml_signal_autoconnect() сканирует файл ***.glade** в поисках сигналов и функций-обработчиков, автоматически связывая сигналы с назначенными им обработчиками, если обработчики являются функциями API *GTK+* (например, эта функция может связать сигнал destroy и функцию-обработчик gtk_main_quit). Неудобство, связанное с использованием glade_xml_signal_autoconnect() (простите за невольный каламбур) заключается в том, что эта функция будет выдавать предупреждающее сообщение всякий раз, когда ей встретится функция-обработчик, не являющаяся частью API *GTK+*. Определенный интерес представляют также функции glade_xml_signal_connect_full() и glade_xml_signal_autoconnect_full (). Первым параметром каждой из этих функций служит, как обычно, указатель на структуру *GladeXML*. Вторым параметром обеих функций должен быть указатель на определенную программистом функцию *GladeXML*ConnectFunc(), которая будет осуществлять связывание сигналов с обработчиками. Таким образом, связывание сигналов и обработчиков как бы делегируется другой функции, не являющейся частью API Glade XML. Такое делегирование может быть полезно при интеграции API Glade XML с другими языками программирования.

В примере из предыдущей статьи в нашем распоряжении имелась удобная функция lookup_widget(), которая позволяла найти указатель на объект визуального элемента по его имени. Эта функция не является частью API *GTK+*, система генерации кода *Glade 2.x* сама создала ее для нас. Рад сообщить вам, что при работе с библиотекой libglade у нас есть возможность использовать похожую функцию glade_xml_get_widget(). Эта функция возвращает указатель на визуальный объект, заданный своим именем. Первым аргументом glade_xml_get_widget() должен быть указатель на структуру *GladeXML*, вторым аргументом – имя искомого объекта. Конечно, для того, чтобы мы могли воспользоваться функцией glade_xml_get_widget() в обработчике сигнала, переменная-указатель на структуру *GladeXML* должна быть в области видимости процедуры-обработчика. Эту проблему можно решить, объявив переменную-указатель на GladeXML как статическую. Этот метод, однако, неудобен, если обработчик сигнала расположен не в том же файле, что и функция main(). В общем случае указатель на структуру *GladeXML* можно получить другим способом. Если у нас есть указатель на один из визуальных элементов, образующих интерфейс (а в обработчике сигнала как минимум один такой указатель есть всегда), мы можем воспользоваться функцией glade_get_widget_tree(). Аргументом функции glade_get_widget_tree() должен быть указатель на объект, представляющий элемент интерфейса, а ее возвращаемым значением будет указатель на структуру *GladeXML*. Вот как, например, можно получить доступ к этой структуре в некотором обработчике on_button_clicked():

```
void on_button_clicked(GtkButton *button)
{
  GladeXML *xml = glade_get_widget_tree(GTK_WIDGET(button));
...
}
```

Материал, рассмотренный в этой, равно как и в четырех предыдущих статьях, не охватывает всего многообразия программирования *GTK+*. Но, я надеюсь, что эти статьи создали нечто вроде стартовой площадки, которая поможет вам вознестись к настоящим высотам, если программирование *GTK+* вас заинтересует. Мы же перейдем к освоению премудростей программирования для основанной на *GTK+* среды GNOME. LXF

Через месяц До свидания, *GTK+*, здравствуй, GNOME! Что бы там ни говорил Линус...