Программирование GNOME —

ЧАСТЬ 6 *GTK+* – это не только инструментарий, это еще и основа сразу для двух рабочих сред! Настало время разобраться с GNOME – **Андрей Боровский** показывает, как создать свое первое GNOME-приложение.

GNOME и *GTK+* соотносятся между собой примерно так же, как KDE и *Qt*, с той разницей, что разработчики открытых сообществ GNOME и *GTK+* всегда лучше взаимодействовали между собой, чем разработчики «соборного» *Qt* и «базарного» KDE.

Если вы стоите перед выбором – использовать ли в своей программе только визуальные компоненты *GTK+*, или же добавить к ним расширения GNOME, соображения в пользу каждого варианта будут примерно такими же, что и при выборе между «чистым» *Qt* и KDE. Компоненты GNOME могут сделать больше (правда, ненамного), чем компоненты *GTK+*, но GNOME – это не просто набор виджетов, а оболочка, и программа, использующая возможности GNOME, сможет работать только на том компьютере, где установлена эта оболочка. В то же время программы, основанные на *GTK+*, могут быть перенесены и в такие среды, где гномы сроду не водились – например, на платформу Microsoft Windows. Впрочем, говоря о том, что нет GNOME для Windows, я не совсем прав: его портирование выполняется в рамках проекта *Cygwin*. Однако нельзя сказать, чтобы CyGNOME был популярен среди пользователей Windows, так что и желающих установить его ради одной вашей программы найдется немного.

В то время как в тандеме *Qt*/KDE практически каждому компоненту *Qt* соответствует свой компонент KDE, компоненты *GTK+* и GNOME практически не дублируют друг друга. К богатому набору *GTK+* GNOME добавляет в основном те компоненты, которые требуются для написания программ, взаимодействующих с оболочкой GNOME. Среда GNOME включает в себя несколько разделяемых библиотек, с которыми компонуются исполняемые файлы приложений GNOME. Важнейшими из этих библиотек являются две – *libgnome* (известная в документации как GNOME Library) и *libgnomeui* (в документации она значится под именем GNOME UI Library). Библиотека *libgnome* экспортирует базовые функции, необходимые для инициализации GNOME-программы, конфигурации программ GNOME и интернационализации. Кроме того, библиотека *libgnome* предоставляет в распоряжение программиста несколько функций для решения таких задач, как, например, отображение встроенной справки и работа со звуком. Библиотека *libgnomeui* экспортирует функции, обеспечивающие работу дополнительных визуальных элементов GNOME.

Наша первая программа для GNOME (файл **openurl.c**) не так уж сильно отличается от простой программы *GTK+*:

› **Рис. 1.**

```
#include <gnome.h>

void button_clicked(GtkWidget * button, gpointer data)
{
    GtkEntry * entry = data;
    GError * error = NULL;
    const char * url = gtk_entry_get_text(entry);
    if (!gnome_url_show(url, &error)) {
      g_print("%s\n", error->message);
      g_error_free(error);
    }
}

gint delete_event(GtkWidget *widget, GdkEvent *event, gpointer data)
{
    gtk_main_quit();
    return FALSE;
}

int main(int argc, char * argv[])
{
    GnomeProgram * gnome_prog;
    GtkWidget * mainwnd;
    GtkWidget * hbox;
    GtkWidget * button;
    GtkWidget * entry;
    gnome_prog = gnome_program_init("openurl", "0.1", LIBGNOMEUI_
MODULE,
        argc, argv, NULL, NULL);
    mainwnd = gnome_app_new ("openurl", "Открыть URL");
    gtk_signal_connect (GTK_OBJECT (mainwnd), "delete_event",
                GTK_SIGNAL_FUNC(delete_event), NULL);
    hbox = gtk_hbox_new (FALSE,5);
    entry = gtk_entry_new();
    gtk_box_pack_start (GTK_BOX(hbox), entry, TRUE, TRUE, 0);
    button = gtk_button_new_with_label("Открыть");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
      GTK_SIGNAL_FUNC (button_clicked), entry);
    gtk_box_pack_start (GTK_BOX(hbox), button, FALSE, FALSE, 0);
    gnome_app_set_contents (GNOME_APP (mainwnd), hbox);
    gtk_widget_show_all(mainwnd);
    gtk_main ();
    return 0;
}
```

Месяц назад Мы завершили изучение «чистого» *GTK+* обсуждением новых возможностей *Glade 3.x*.

первые шаги

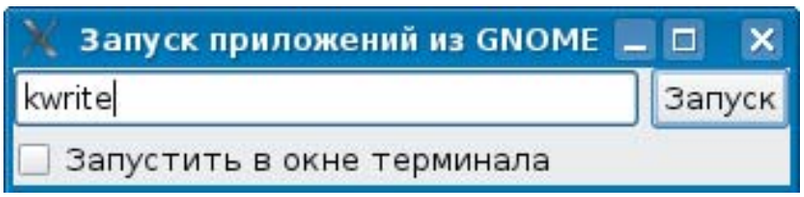
Прежде чем разбирать исходный текст программы, полезно узнать, что она делает. Окно программы *openurl* (**Рис. 1**) содержит строку ввода и кнопку. Если ввести в строке адрес URL, а затем щелкнуть кнопку мышью, программа запустит соответствующий внешний интернет-клиент и передаст ему введенный URL. Под «соответствующим Интернет-клиентом» в данном случае понимается клиентская программа, которая назначена в вашей системе для обработки указанного в URL интернет-протокола (HTTP, FTP и т.д.) по умолчанию. Информацию о том, какую именно программу нужно запустить для обработки той или иной ссылки URL, программа *openurl* получает из настроек системы, так что если программа не может открыть ссылку какого-то типа, проверьте, назначен ли для соответствующего протокола клиент по умолчанию. Пошаговый разбор программы *openurl* мы начнем с заголовочного файла. Хотя наша программа использует функции библиотек *libgnome*, *libgnomeui* и, конечно, *GTK+*, нам достаточно включить в исходный текст один заголовочный файл – **gnome.h**. В результате станут доступны прототипы функций всех перечисленных интерфейсов. Пропустим пока обработчики сигналов и рассмотрим функцию main(). Работа главной функции программы начинается с вызова функции gnome_program_init(). Она инициализирует библиотеки GNOME (и *GTK+*), с которыми будет взаимодействовать наша программа. Первым аргументом функции gnome_program_init() является идентификатор приложения – это может быть просто строка с именем исполняемого файла программы. Вторым аргументом должна быть строка с версией приложения. Помимо прочего, функция gnome_program_init() загружает информацию о модулях GNOME, требуемых приложению. Для того, чтобы сообщить системе какие модули нам нужны, достаточно указать в качестве третьего параметра макрос-константу Libgnomeui_MODULE. В качестве четвертого и пятого параметров функции gnome_program_init() передаются параметры argc и argv, полученные функцией main(). Остальные параметры gnome_program_init() (а их может быть много) связаны с обработкой ключей командной строки. Мы не собираемся запускать нашу программу с какими-либо дополнительными ключами и просто передаем два значения **NULL** для того, чтобы функция gnome_program_init() была довольна. В предыдущих версиях GNOME для инициализации программы использовалась функция gnome_init(). И хотя эта функция до сих пор присутствует в интерфейсе GNOME, использовать ее в новых программах категорически не рекомендуется. По моим наблюдением, использование gnome_init() с последними версиями GNOME может нарушить стабильность работы всей графической среды. Функция gnome_program_init() возвращает указатель на объект GnomeProgram. Он содержит много полезной информации о запускаемой GNOME-программе, но обращаться к нему мы пока не будем.

После того, как все нужные нам библиотеки GNOME инициализированы, мы можем приступить к созданию интерфейса программы. Конструирование интерфейса мы начнем с главного окна, которое создается функцией gnome_app_new(). Первым аргументом этой функции должно быть имя программы, вторым аргументом – заголовок главного окна. Функция возвращает указатель на объект GnomeApp, который, вопреки своему названию, является всего лишь улучшенным вариантом (и потомком) объекта GtkWindow. Назначение обработчика события delete_event, создание контейнера для горизонтальной упаковки дочерних элементов, создание и упаковка самих элементов GtkEntry (строка ввода) и GtkButton (кнопка), так же как и назначение обработчика сигнала clicked кнопки, должны быть вам хорошо знакомы, так что мы их пропустим. Функция gnome_app_set_contents() позволяет указать визуальный элемент, который будет отвечать за содержимое рабочей области главного окна программы. Первым аргументом функции gnome_app_set_contents() должен быть указатель на окно GnomeApp, вторым – указатель на его дочерний элемент, управляющий рабочей областью(в качестве такового в нашей программе выступает объект-контейнер hbox).

Функция gtk_widget_show_all() является частью интерфейса *GTK+*, но мы с ней раньше не встречались, поэтому опишем ее подробнее. До сих пор мы делали каждый элемент управления программы видимым с помощью отдельного вызова gtk_widget_show(). Функция gtk_widget_show_all() позволяет «придать видимость» всем визуальным элементам программы за один вызов. Цикл обработки сообщений программы GNOME запускается с помощью функции gtk_main(), предоставляемой интерфейсом *GTK+*.

Перейдем к обработчику сигнала clicked кнопки button – функции button_clicked(). В качестве дополнительного параметра функции-обработчику передается указатель на объект GtkEntry, из которого он должен получить текст адреса URL. Вся «магическая работа» по запуску внешней программы для обработки URL выполняется функцией gnome_url_show(). Ее первым параметром должна быть строка URL, второй аргумент представляет собой указатель на указатель на структуру GError. Если в процессе обработки URL функцией gnome_url_show() возникла ошибка, функция создаст экземпляр структуры GError, и вернет указатель на него во втором параметре. При этом функция также вернет значение **FALSE** как результат вызова. Мы проверяем значение, возвращаемое gnome_url_show(), и в случае возникновения ошибки распечатываем сообщение о ней (текст сообщения содержится в поле message структуры GError). Поскольку при возникновении ошибки каждый раз создается новый экземпляр структуры GError, мы должны удалять его с помощью функции g_error_free(), чтобы не заполнять оперативную память мусором. Обратите внимание, что функция gnome_url_show() следит за тем, чтобы переданная ей переменная-указатель на объект GError имела значение **NULL**. В противном случае функция может, ни много ни мало, досрочно завершить работу программы.

Прежде чем вы начнете проверять, как работает программа openurl, необходимо сказать несколько слов о том, как функция gnome_url_show() обрабатывает переданные ей URL-адреса. Для того, чтобы выбрать внешнюю программу для открытия того или иного URL, функция должна «знать» интернет-протокол, который он использует. Информацию о протоколе функция получает из префикса URL (**http://**, **ftp://** и т.д.), поэтому, вводя URL в строке ввода программы, вы обязательно должны указывать префикс (если вы его не укажете, gnome_url_show() вернет сообщение об ошибке). Отметим также, что ошибки, о которых может сообщить функция gnome_url_show(), относятся исключительно к синтаксису адреса URL, но не к ресурсу, на который она ссылается. Как только функция находит приложение, способное открыть данный URL, она запускает ее, передавая URL в качестве параметра командной строки и возвращает управление нашей программе. Вообще говоря, у нас нет возможности проследить, чем закончилась »



› Рис. 2.

» работа программы, которую запустила функция gnome_url_show().

Компиляция и сборка программы для GNOME требуют дополнительных приготовлений. Лучше всего воспользоваться простым make-файлом. Рассмотрим фрагмент из make-файла, который вы найдете на диске (он собирает все программы-примеры для этой статьи, а для того, чтобы собрать программу *openurl*, вы можете просто скомандовать “make openurl”):

```
CFLAGS =-g -Wall `pkg-config --cflags libgnome-2.0 libgnomeui-2.0`
LDFLAGS=`pkg-config --libs libgnome-2.0 libgnomeui-2.0`
```

Для генерации ключей компилятора и компоновщика мы вызываем утилиту *pkg-config*, указав ей пакеты *libgnome-2.0* и *libgnomeui-2.0*. Теперь программу можно и скомпилировать.

Открыть в терминале?

Запуск внешних Интернет-клиентов – это далеко не все, на что способны программы GNOME. В качестве примера более широких возможностей мы рассмотрим приложение, способное запустить произвольно заданную программу. Наша вторая программа GNOME (ее исходный текст вы найдете в файле **gnomeexec.c**) похожа на первую (**Рис. 2**). Введите имя программы, которую вы хотите запустить, и нажмите кнопку Запуск. Если вы собираетесь работать с консольной программой, можете установить флажок Запустить в окне терминала (в этом случае при запуске программы будет открыто окно терминала, используемого GNOME по умолчанию).

Главная проблема, с которой сталкивается программист, пишущий Unix-программу, которая должна запускать другие программы – это проблема файловых дескрипторов. По умолчанию, дочерний процесс наследует все дескрипторы, открытые родительским процессом. В графической системе GNOME, в которой каждая программа открывает множество служебных дескрипторов, в том числе для связи с сервером X и CORBA, разделение этих дескрипторов с новым процессом может вызвать проблемы у обеих программ. Запуская новую программу, программист должен быть уверен, что она не унаследует дескрипторы, которые должны принадлежать исключительно родительской программе. Если вы думаете, что далее я приведу пример закрытия дескрипторов (как в статьях по программированию Unix API), то ошибаетесь. Дело в том, что интерфейс программирования GNOME предоставляет в распоряжение программиста целое семейство функций, предназначенных для запуска из программы GNOME других программ. Все проблемы, связанные с дескрипторами, эти функции решают за нас.

Всего в нашем распоряжении находится восемь функций интерфейса GNOME, позволяющих запустить внешнюю программу. Мы воспользуемся двумя – gnome_execute_shell() и gnome_execute_terminal_shell(). Эти функции позволяют запускать внешние программы с теми переменными окружения, которые установлены в оболочке, используемой пользователем по умолчанию. Функция gnome_execute_shell() просто запускает программу на выполнение, а функция gnome_execute_terminal_shell() сначала открывает окно терминала и запускает программу в нем. Обе функции принимают два аргумента – рабочую директорию программы и строку запуска. Если в качестве первого аргумента функциям передать значение **NULL**, новое приложение унаследует рабочий каталог программы-родителя.

Рассмотрим обработчик сигнала clicked, определенный в программе *gnomeexec*. Именно в нем и выполняется запуск внешней программы:

```
GtkWidget * entry;
GtkWidget * check_button;

void button_clicked(GtkWidget * button, gpointer data)
{
    const char * cmdline = gtk_entry_get_text(GTK_ENTRY(entry));
    if (gtk_toggle_button_get_active(GTK_TOGGLE_BUTTON(check_
button)))
                gnome_execute_terminal_shell(NULL, cmdline);
        else
                gnome_execute_shell(NULL, cmdline);
}
```

В программе *openurl* обработчику требовались данные только одного элемента управления – строки ввода. В программе *gnomeexec* обработчик должен, помимо строки запуска программы, получить данные о состоянии флажка, который определяет, следует ли запускать программу в окне терминала. В качестве строки ввода мы используем все тот же объект GtkEntry. Кнопка-флажок реализована в *GTK+* с помощью объекта GtkCheckButton. Таким образом, обработчик сигнала clicked должен получить информацию сразу о двух объектах. Для того, чтобы они были доступны обработчику, мы объявляем переменные entry и check_button глобально, в области видимости обработчика и функции main().

Из объекта entry обработчик извлекает строку запуска программы. Какая именно из функций запуска внешней программы будет использована, зависит от того, установлен ли флажок check_button. Его состояние проверяется с помощью функции gtk_toggle_button_get_active() (объект GtkCheckButton является потомком объекта GtkToggleButton, который служит предком всех кнопок-переключателей). Если функция gtk_toggle_button_get_active() возвращает TRUE, значит, флажок установлен и мы запускаем внешнюю программу в окне терминала с помощью функции gnome_execute_terminal_shell(). Если функция gtk_toggle_button_get_active() возвращает значение **FALSE**, для запуска программы используется функция gnome_execute_shell().

В программе *gnomeexec* есть еще несколько строчек кода, заслуживающих вашего внимания. Мы привыкли к тому, что если в окне программы есть «главная» кнопка, выполняющая некое действие, то нажатие клавиши Enter равносильно щелчку мышью по этой кнопке. Работая с программой *openurl*, вы наверняка заметили, что эта программа игнорирует клавишу Enter, так что кнопку Открыть приходится щелкать мышью. В окне, где сначала нужно ввести текст с клавиатуры, необходимость переключаться на мышь для щелчка по кнопке неудобна вдвойне. В программе *gnomeexec* мы исправим этот недостаток, но прежде – небольшое лирическое отступление. Мы любим программирование за то, что оно позволяет нам реализовать наши творческие замыслы, и ненавидим его за разные неожиданные препятствия, которые оно возводит на нашем пути. Как заставить кнопку реагировать на нажатие Enter? Знатоки графических интерфейсов могут ожидать, что существует какая-нибудь функция gtk_button_set_default(), которая делает заданную кнопку «кнопкой по умолчанию». Ничего подобного в *GTK+ API* нет. Чтобы заставить кнопку реагировать на клавишу Enter, нам придется вызвать две функции и один макрос. Прежде всего, если в окне есть объект GtkEntry (а в окне *gnomeexec* он есть), по умолчанию он перехватывает все события, связанные с клавиатурой. Для того, чтобы объект GtkEntry мог передать обработку нажатия на клавишу Enter другим элементам управления окна, мы должны вызвать функцию gtk_entry_set_activates_default(). Далее, оказывается, что кнопка GtkButton по умолчанию не относится к числу объектов, способных (простите за каламбур) обрабатывать нажатие Enter по умолчанию. Мы, однако, можем наделить кнопку GtkButton этой способностью, если вызовем макрос GTK_WIDGET_SET_FLAGS() с константой GTK_CAN_DEFAULT:

```
GTK_WIDGET_SET_FLAGS (button, GTK_CAN_DEFAULT);
```

Макрос GTK_WIDGET_SET_FLAGS() позволяет устанавливать различные флаги, влияющие на свойства и поведение визуальных элементов. Только теперь мы можем вызвать функцию gtk_widget_grab_default(), которая заставит кнопку реагировать на нажатие Enter как на щелчок мышью. Справедливости ради стоит отметить, что при использовании окна, создаваемого объектом GtkDialog, определение «кнопок

по умолчанию» упрощается. После того, как поведение кнопки приведено в соответствие с правилами хорошего интерфейса, мы можем собрать программу, скомандовав

```
make gnomeexec
```

А как это будет по-русски?

Интернационализация приложений GNOME выполняется почти так же, как и приложений *GTK+* (в связи с чем рекомендую перечитать LXF87/88), однако есть и небольшие отличия. В процессе интернационализации приложений *GTK+* нам требовалось самим определить несколько макросов. В заголовочных файлах GNOME эти макросы уже определены, и повторное их определение приведет к выдаче компилятором предупреждающих сообщений. В качестве примера выполним интернационализацию программы *gnomeexec* (этот вариант программы вы найдете в файле **gnomeexec-i18n.c**). Никаких дополнительных заголовочных файлов в текст программы включать не требуется – достаточно уже имеющегося **gnome.h**. Наши действия по подготовке интерфейса программы к переводу на другие языки сводятся к трем шагам. Во-первых, в функцию main() мы добавляем вызовы трех макросов:

```
bindtextdomain (GETTEXT_PACKAGE, LOCALEDIR);
bind_textdomain_codeset (GETTEXT_PACKAGE, “UTF-8”);
textdomain (GETTEXT_PACKAGE);
```

То же самое мы делали в процессе интернационализации программы *GTK+*. Макросы располагаются в самом начале функции main(), еще до вызова функции gnome_program_init().

Второй шаг заключается в пометке всех строк, предназначенных для перевода, макросом _(). В *GTK+* мы определяли этот макрос сами, в GNOME он уже определен.

Нашим третьим шагом должно быть определение констант-макросов GETTEXT_PACKAGE и LOCALEDIR. Поскольку для компиляции программы мы используем make-файлы, будет удобно разместить объявления этих макросов в нем. В результате исходный текст нашей программы вообще не будет содержать никаких определений макросов. В make-файл для сборки программы *gnomeexec-i18n* мы добавим строку

```
CFLAGS =-g -Wall `pkg-config --cflags libgnome-2.0 libgnomeui-2.0` \\
-DLOCALEDIR=\””./locale”\” -DGETTEXT_PACKAGE=\””gnomeexec-i18n”\” \\
-DENABLE_NLS
```

Помимо двух вышеуказанных констант мы включаем директиву условной компиляции ENABLE_NLS, которая делает доступными все макросы и функции интернационализации, объявленные в заголовочных файлах, включенных в **gnome.h**. Теперь текст интерфейса программы *gnomeexec-i18n* готов к переводу. Остальные действия, связанные с представлением его интерфейса на других языках, выполняются точно так же как и при работе с *GTK+*. С помощью утилиты *xgettext* мы создаем каталог строк, предназначенных для перевода, и переводим строки из каталога, используя любую подходящую утилиту (например, *KBabel*), затем, при помощи утилиты *msgfmt*, компилируем переведенный каталог в двоичный файл перевода и размещаем его в директории, указанной в константе LOCALEDIR. Серьезные приложения GNOME, предназначенные для всех пользователей системы, размещают свои файлы перевода в директории **/opt/gnome/share/locale**, мы же не претендуем на такую честь и помещаем файл перевода нашей программы в локальном каталоге.

Я признаю, что программы из этой статьи отличались довольно незамысловатым интерфейсом. Все дело в том, что мне лень вручную громоздить контейнеры друг на друга (даже если эти контейнеры – всего лишь объекты *GTK+*). Следующая статья этой серии расскажет вам, как создавать роскошные интерфейсы GNOME с помощью программ *Glade* и *Anjuta*. LXF

» **Через месяц** Визуальное проектирование интерфейсов GNOME – и загадочная Anjuta.