Погружение

ЧАСТЬ 8 Мы уже успели замочить ножки в океане возможностей, которые предоставляет GNOME – и сейчас самое время зайти поглубже. **Андрей Боровский** проверяет, теплая ли водичка.

В прошлый раз мы начали работу над демо-приложением GNOME – текстовым редактором. До сих пор в этой программе собственно «гномовского» было не так уж много: только код инициализации. На этот раз мы рассмотрим некоторые функции, относящиеся исключительно к GNOME; но сначала – немного визуального программирования.

Редактирование меню

Заготовка главного меню нашей программы содержит пункт Вид (View), однако само это меню пустое. Добавим в меню Вид новую команду Шрифт..., позволяющую выбрать шрифт для отображения текста. Щелкните правой кнопкой мыши по строке главного меню в окне формы приложения и в открывшемся контекстном меню выберите команду Правка меню... (**Рис. 1**).

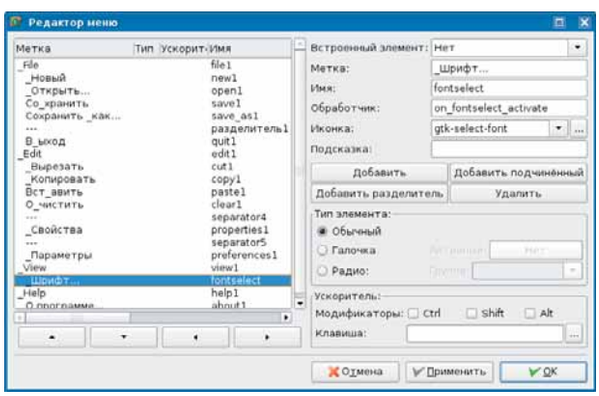


Рис. 1. Редактор меню *Glade*.

Список названий пунктов меню отражает существующие между ними иерархические отношения. Сдвиг строки вправо означает переход к следующему уровню вложенности меню. С помощью кнопок Добавить и Добавить подчиненный в меню можно добавить новый пункт. При добавлении нового подчиненного пункта необходимо указать для него родительский пункт меню. Если мы потом передумаем, то с помощью клавиш навигации сможем переместить уже созданный пункт в другую группу или на другой уровень вложенности.

Раскрывающийся список Встроенный элемент позволяет присвоить новому пункту меню свойства одного из стандартных элементов меню (эти встроенные элементы используют пункты меню, созданные автоматически). Строка ввода Метка позволяет указать название пункта меню. Нижний дефис в названии пункта меню отмечает подчеркнутый символ, который нужно вводить в сочетании с Alt для быстрого доступа к этому пункту. Строка ввода имя позволяет указать имя объекта GtkImageMenuItem, соответствующего пункту меню. В строке ввода Обработчик указывается имя функции-обработчика сигнала activate, посылаемого командой меню. Комбинированный раскрывающийся список Иконка позволяет указать пиктограмму, которая будет отображаться рядом с командой меню. Можно загрузить собственную пиктограмму из файла либо использовать одну из стандартных пиктограмм, установленных в системе.

Для создаваемого нами пункта меню Шрифт... выберем стандартную пиктограмму *gtk-select-font*. Остальные элементы редактора меню нам сейчас не интересны, так что мы их опустим (читателю, как всегда, рекомендуется обратиться к документации). Щелкните кнопку OK. Теперь у нас есть новая команда меню, для которой нужно написать обработчик (функция on_fontselect_activate()). Для выбора шрифта используйте уже знакомое нам диалоговое окно GtkFontSelectionDialog. Полный код обработчика вы найдете на диске, мы на нем останавливаться не будем.

Буфер обмена GTK+ и GNOME

Интерфейс буфера обмена в *GTK+* основан на структуре GtkClipboard. Мы получаем доступ к этой структуре с помощью функции gtk_clipboard_get(). Первый вызов gtk_clipboard_get() создает экземпляр структуры, которая переходит под управление *GTK+* (так что мы не должны удалять ее явным образом). При последующих вызовах gtk_clipboard_get() возвращает указатель на существующую структуру GtkClipboard.

При работе с X Window следует учесть, что система поддерживает как минимум два буфера обмена (теоретически их может быть и больше). Если вы используете команду меню Правка | Копировать, данные обычно попадают в буфер обмена, связанный с атомом CLIPBOARD. В то же время любые данные, выделенные мышью в окне X-программы, заносятся в буфер обмена, связанный с атомом PRIMARY (вставка данных из этого буфера обычно выполняется с помощью щелчка средней кнопкой мыши). Вообще говоря, X-программы вольны интерпретировать оба буфера так, как им заблагорассудится. На практике это иногда приводит к тому, что результаты вставки данных из буфера обмена оказываются несколько неожиданными.

Месяц назад Мы изучали среду *Anjuta* и невзначай написали простой текстовый редактор.

в GNOME

Интерфейс буфера обмена *GTK+* позволяет работать со всеми буферами обмена *X*. *GTK+* поддерживает размещение данных в буферах обмена в нескольких форматах и отложенную запись.

Организовать передачу данных между буфером обмена и экземпляром GtkTextBuffer очень просто. Ниже приводится текст обработчика сигнала меню copy1_activate нашего текстового редактора (этот сигнал генерирует команда Правка | Копировать).

```
void on_copy1_activate (GtkMenuItem * menuitem, gpointer user_data)
{
char * atom = gdk_atom_name("CLIPBOARD");
GtkClipboard * cb = gtk_clipboard_get(atom);
g_free(atom);
gtk_text_buffer_copy_clipboard(textbuffer, cb);
}
```

Вызывая gtk_clipboard_get(), мы передаем этой функции X-атом, идентифицирующий буфер обмена, который мы хотим открыть, и получаем строку с именем атома с помощью функции gdk_atom_name(). Функция gtk_text_buffer_copy_clipboard() копирует в буфер обмена строку текста, выделенную в текстовом буфере GtkTextBuffer (программно или с помощью интерфейса пользователя). Первым параметром функции, копирующей данные, должен быть указатель на объект GtkTextBuffer, вторым параметром – указатель на объект GtkClipboard. Для вставки данных из буфера обмена в текстовый буфер применяется функция gtk_text_buffer_paste_clipboard(), которой, помимо прочего, следует передать итератор, указывающий, где именно в тексте должна быть вставлена строка из буфера обмена (при использовании текстового буфера вместе с GtkTextView вместо итератора функции можно передать NULL).

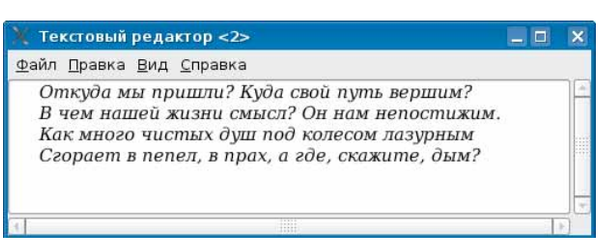
Разумеется, у буфера обмена есть и собственный API, который можно применять независимо от других компонентов *GTK+*/GNOME. Для передачи в буфер обмена строки текста в общем случае используется функция gtk_clipboard_set_text(). Первым параметром этой функции должен быть указатель на объект GtkClipboard. Второй и третий параметры, соответственно, строка текста с нулевым конечным символом и длина строки в байтах (напомню, что в *GTK+* и GNOME по умолчанию используется кодировка UTF-8). Для передачи в буфер обмена растрового графического объекта служит функция gtk_clipboard_set_image(). У этой функции всего два параметра: указатель на объект GtkClipboard и указатель на объект GdkPixBuf, содержащий данные растрового изображения.

Если вы хотите поместить в буфер обмена данные в нескольких форматах или реализовать отложенную запись, все становится более сложным. В этом случае ваша программа должна объявить системе, что она готова передать в буфер обмена данные в определенных форматах и зарегистрировать функции обратного вызова, которые будут выполнять их фактическую обработку. Когда данные понадобятся, эти функции будут вызваны системой. Рассмотрим пример – фрагмент программы, копирующей в буфер строку текста:

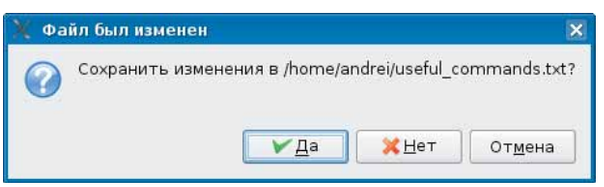
```
void cb_get_func(GtkClipboard * clipboard, GtkSelectionData * selection_
data,
guint info, gpointer user_data)
{
char * str = "Данные для буфера обмена";
gtk_selection_data_set_text(selection_data, str, strlen(str));
}

void cb_clear_func(GtkClipboard * clipboard, gpointer user_data)
{
// Nothing to do
}

void on_copy1_activate (GtkMenuItem * menuitem, gpointer user_data)
{
static const GtkTargetEntry targets[] =
 { { "UTF8_STRING", 0, GDK_TARGET_STRING } };
char * atom = gdk_atom_name("CLIPBOARD");
GtkClipboard * cb = gtk_clipboard_get(atom);
g_free(atom);
gtk_clipboard_set_with_data(cb, targets, 1, cb_get_func, cb_clear_func,
NULL);
}
```

Процесс записи данных в буфер обмена инициирует все тот же обработчик on_copy1_activate(). Регистрация форматов данных в буфере обмена начинается с создания списка форматов в виде массива структур GtkTargetEntry. Мы немного упростили себе жизнь за счет того, что сконструировали этот массив (который в нашем случае состоит из одного элемента) статически. В сложных программах список форматов, в которых копируются данные, может быть заранее неизвестен, и тогда его придется создавать динамически, с помощью интерфейса структуры GtkTargetList.

Получив указатель на объект GtkClipboard, мы вызываем функцию gtk_clipboard_set_with_data(). Эта функция регистрирует список фор-



› Рис. 2.



› Рис. 3.

» матов данных, которые программа может передать в буфер обмена (третий параметр функции – количество элементов в массиве targets), и передает системе адреса двух функций обратного вызова. Первая функция (cb_get_func()) будет вызвана для того, чтобы передать в буфер обмена данные в запрошенном формате. Вторая функция (cb_clear_func()) должна оповестить программу о том, что содержимое буфера обмена было перезаписано другим приложением (и ресурсы, выделенные нашей программой для буфера обмена, можно освободить). Последний параметр функции gtk_clipboard_set_with_data() – указатель, который будет передан функциям обратного вызова: его можно использовать для передачи функциям произвольных данных. Поскольку в нашем примере мы не выделяли динамически никаких ресурсов для работы с буфером, в функции cb_clear_func() не нужно ничего и высвобождать.

Рассмотрим подробнее функцию cb_get_func(). Помимо прочего, этой функции передается указатель на объект GtkSelectionData. Данные записываются в буфер с помощью функций, связанных с этим объектом. Для записи данных в текстовом формате используется функция gtk_selection_data_set_text(). Запись в буфер обмена растровой картинки выполняется функцией gtk_selection_data_set_pixbuf(), а для передачи списка ссылок на ресурсы (сетевые или файловые) применяется функция gtk_selection_data_set_uris(). Обратите внимание на параметр info функции cb_get_func(). Этот параметр содержит идентификатор формата, в котором система желает получить данные. Поскольку мы зарегистрировали только один формат, значение info можно не проверять. В параметре user_data функциям обратного вызова передается указатель на дополнительные данные. Зарегистрировать форматы данных и функции обратного вызова можно также с помощью функции gtk_clipboard_set_with_owner(), которая отличается от gtk_clipboard_set_with_data() только тем, что в качестве дополнительного параметра функциям обратного вызова передается указатель GObject *, а не gpointer.

Для вставки данных из буфера обмена также реализованы два интерфейса. Вы можете зарегистрировать функцию, которая будет вызвана системой при появлении в буфере обмена данных. Регистрация функций обратного вызова для вставки данных выполняется с помощью функций семейства gtk_clipboard_request_XXX(). Хотя функция обратного вызова регистрируется gtk_clipboard_request_XXX() для приема определенного типа данных, она будет вызвана в ответ на передачу в буфер любых данных. Если в буфере обмена появились данные в неподходящем формате, функции обратного вызова будет передано значение **NULL**. Если вы не хотите иметь дело с обратными вызовами, воспользуйтесь функциями gtk_clipboard_wait_for_text() и gtk_clipboard_wait_for_image() и им подобными. Эти функции сами возвращают данные из буфера обмена и не требуют использования обратных вызовов.

Виртуальная файловая система GNOME

Как и KDE, GNOME стремится объединить файловые ресурсы сети и локальные файловые ресурсы в единое файловое пространство. С этой целью в GNOME, как и в KDE, реализован дополнительный уровень абстракции поверх файловой системы. Виртуальная файловая система GNOME (GNOME VFS) предоставляет единый интерфейс программирования для доступа к самым разным ресурсам, которые могут быть представлены в виде файлов, а также включает множество функций, упрощающих работу с файлами.

Рассмотрим функцию open_file() нашего текстового редактора (листинг сокращен):

```
void open_file(char * file_name)
{
GnomeVFSHandle * file_handle;
GnomeVFSFileInfo * file_info;
GnomeVFSFileSize bytes_read;
gpointer buf;
if (gnome_vfs_open(&file_handle, file_name, GNOME_VFS_OPEN_READ)
!=
 GNOME_VFS_OK) {
// Сообщить пользователю об ошибке.
}
strncpy(filename, file_name, 256);
gtk_text_buffer_set_text(textbuffer, "", 0);
file_info = gnome_vfs_file_info_new();
gnome_vfs_get_file_info_from_handle(file_handle, file_info,
GNOME_VFS_FILE_INFO_DEFAULT);
buf = g_malloc(file_info.size);
gnome_vfs_read(file_handle, buf, file_info.size, &bytes_read);
gtk_text_buffer_set_text(textbuffer, buf, bytes_read);
g_free(buf);
gnome_vfs_file_info_unref(file_info);
gnome_vfs_close(file_handle);
gtk_text_buffer_set_modified(textbuffer, FALSE);
}
```

Функции GNOME VFS API объявлены в заголовочном файле **libgnomevfs/gnome-vfs.h** (сама библиотека *libgnomevfs* включена в проект приложения GNOME, созданный в *Anjuta*, по умолчанию). Систему GNOME VFS следует инициализировать при помощи функции gnome_vfs_init() (эту функцию достаточно вызвать один раз). По окончании работы с GNOME VFS мы вызываем функцию gnome_vfs_shutdown(). Для работы с файлами GNOME VFS предоставляет набор функций, похожих на функции POSIX.

Файл открывается с помощью функции gnome_vfs_open(). В отличие от функций POSIX, gnome_vfs_open() не возвращает дескриптор файла: функция возвращает константу, указывающую статус завершения операции. Роль дескриптора файла при работе с функциями GNOME VFS выполняет значение типа VFSHandle (адрес переменной, в которую будет записан указатель на переменную типа VFSHandle, передается функции gnome_vfs_open() в первом параметре). Второй параметр gnome_vfs_open() – имя открываемого файла, а в третьем параметре функции передаются флаги, описывающие то, для чего открывается файл (GNOME_VFS_OPEN_READ – для чтения, GNOME_VFS_OPEN_WRITE – для записи, и т.д.).

Вы можете преобразовать дескриптор файла в VFSHandle с помощью функции gnome_vfs_open_fd(). Функция gnome_vfs_read() предназначена для чтения данных из файла. Как и gnome_vfs_open(), она возвращает информацию о статусе выполняемой операции. Все остальные результаты передаются через параметры. Первый параметр gnome_vfs_read() – указатель на VFSHandle файла, из которого нужно читать данные. Далее следуют указатель на буфер для данных, размер буфера и указатель на переменную, в которую функция запишет, сколько байтов ей удалось прочитать. Для записи данных используется функция gnome_vfs_write() с тем же списком параметров, что и у gnome_vfs_read().

Закончив работу с файлом, мы закрываем его, используя функцию gnome_vfs_close(). Получить информацию об открытом файле (по крайней мере, о файле локальной файловой системы) можно с помощью функции gnome_vfs_get_file_info_from_handle(). Ее первым параметром должен быть указатель на идентификатор открытого файла VFSHandle, вторым параметром – указатель на экземпляр структуры GnomeVFSFileInfo, в который будут записаны сведения о файле, а третьим параметром – комбинация флагов, указывающих, какие именно данные мы хотим получить.

Экземпляр GnomeVFSFileInfo создается с помощью функции gnome_vfs_file_info_new(). Функция gnome_vfs_file_info_unref() уменьшает внутренний счетчик ссылок структуры GnomeVFSFileInfo и удаляет структуру при обнулении счетчика. Структура GnomeVFSFileInfo содержит множество полей, но не все они заполняются по умолчанию (это сделано, в том числе, ради быстродействия). Специальные флаги заставляют функцию gnome_vfs_get_file_info_from_handle() возвращать дополнительные сведения о файле. Помимо функции gnome_vfs_get_file_info_from_handle(), есть еще функция gnome_vfs_get_file_info(), позволяющая получить информацию о файле, заданным своим именем. Для работы со структурой GnomeVFSFileInfo определено несколь-

ко дополнительных функций, более или менее подробное описание которых можно найти в документации GNOME.

Чтобы создать новый или перезаписать уже существующий файл, применяется функция gnome_vfs_create(). Первые три параметра у этой функции те же, что и у gnome_vfs_open(). В четвертом параметре функции передается значение типа gboolean, указывающее, должна ли функция перезаписать файл, если он уже существует. В пятом параметре следует передать маску разрешений для создаваемого файла. У функций GNOME VFS, работающих с именем файла в текстовом формате, есть аналоги с окончанием _uri. Этим функциям вместо строки с именем файла передается указатель на структуру GnomeVFSURI, которая упрощает работу с универсальными идентификаторами ресурсов.

В чем же преимущество функций GNOME VFS API перед функциями POSIX? Предположим, нам нужно загрузить в с Web-сайта документ **www.somesite.com/somefile.txt**. Нет ничего проще – открываем файл для чтения:

```
gnome_vfs_open(&file_handle, “http://www.somesite.com/somefile.txt”,
GNOME_VFS_OPEN_READ);
```

Префикс **http://** надо указывать обязательно, иначе система не узнает, какой модуль виртуальной файловой системы должен использоваться для загрузки ресурса. Дальше данные можно считывать с помощью gnome_vfs_read(). Загрузить файл на FTP-сайт тоже очень просто. Создаем или перезаписываем файл, например:

```
gnome_vfs_create(&file_handle, “ftp://user:password@somesite.com/
home/user/somefile.txt”, GNOME_VFS_OPEN_WRITE, FALSE, 0666);
```

Затем данные записываются в новый файл с помощью gnome_vfs_write() (для работы с идентификаторами FTP-ресурсов лучше использовать структуру GnomeVFSURI и соответствующие функции). Благодаря GNOME VFS мы получаем возможность использовать единый интерфейс и одну программную логику для доступа к самым разным ресурсам. Отличия могут быть связаны только с быстродействием. Если программа ориентирована прежде всего на работу с локальными файлами, чаще используют блокирующие функции в главном потоке (как и мы в рассмотренном примере). Если программа нацелена на работу с ресурсами Глобальной сети, лучше использовать потоки GNOME или асинхронные функции ввода/вывода GNOME VFS.

Единый интерфейс работы с разными ресурсами реализуют модули GNOME VFS. Модули GNOME VFS поддерживают работу с HTTP, FTP, защиту данных с помощью SSL и низкоуровневую работу с сокетами. Вообще говоря, модули GNOME VFS позволяют представить в виде файловых систем множество самых разных источников данных, а если нужного именно вам модуля в системе нет, вы можете написать его сами – это не так уж и трудно. GNOME VFS позволяет конвейеризовать консольные утилиты прямо в строке идентификатора ресурса с помощью разделителя #. Таким образом, можно приказать системе, чтобы она загрузила с web-сайта архив **tar.gz**, распаковала его на лету и передала программе данные для чтения.

С помощью GNOME VFS можно работать с MIME-типами, осуществлять мониторинг изменения файлов, директорий и монтирования файловых систем и многое-многое другое.

Есть у GNOME VFS и набор специальных функций для работы с директориями. Рассмотрим фрагмент консольной программы (графический интерфейс? Зачем нам графический интерфейс?) *scandir*. Функция scan_directory(), определенная в этой программе, сканирует директорию, имя которой передано ей в ее единственном параметре, и распечатывает данные о содержащихся в этой директории файлах. Рассмотрим сокращенный листинг этой функции (полный вариант, как всегда, на диске).

```
int scan_directory(char * dir_name)
{
GnomeVFSDirectoryHandle * handle;
GnomeVFSFileInfo * file_info;
GnomeVFSResult res;
if ((res = gnome_vfs_directory_open(&handle, dir_name,  GNOME_VFS_
FILE_INFO_FOLLOW_LINKS)) != GNOME_VFS_OK) {
g_print(“failed to open the directory: %s\n”,
gnome_vfs_result_to_string(res));
return res;
}
file_info = gnome_vfs_file_info_new();
while ((res = gnome_vfs_directory_read_next(handle, file_info)) ==
GNOME_VFS_OK) {
g_print(“name: %s\n”, file_info->name);
g_print(“type: “);
switch(file_info->type) {
case GNOME_VFS_FILE_TYPE_REGULAR:
g_print(“file\n”);
break;
...
case GNOME_VFS_FILE_TYPE_UNKNOWN:
g_print(“unknown\n”);
break;
}
g_print(“size: %lli\n\n”, file_info->size);
gnome_vfs_file_info_clear(file_info);
}
gnome_vfs_directory_close(handle);
gnome_vfs_file_info_unref(file_info);
if (res != GNOME_VFS_ERROR_EOF) {
g_print(“Error reading the directory: %s\n”,
gnome_vfs_result_to_string(res));
return res;
}
    return 0;
}
```

Мы открываем директорию для чтения с помощью функции gnome_vfs_directory_open(). Открытая директория идентифицируется в GNOME VFS значением типа GnomeVFSDirectoryHandle (адрес переменной, которой должен быть присвоен указатель на это значение, передается как первый параметр функции gnome_vfs_directory_open()). Во втором параметре функции передается имя директории, которую вы хотите открыть, а в третьем – набор флагов, определяющих, какие данные о дочерних элементах директории вы хотите получать (эти флаги – того же типа, что и флаги функции vfs_get_file_info()).

Как и все функции GNOME VFS API, gnome_vfs_directory_open() возвращает численный код, свидетельствующий о том, была ли она выполнена и какие возникли ошибки. Мы можем получить соответствующую этому коду строку пояснения на английском языке с помощью функции gnome_vfs_result_to_string(). Когда работа с директорией закончена, она закрывается с помощью функции gnome_vfs_directory_close(). Для чтения содержимого директории предназначена функция gnome_vfs_directory_read_next(), которая записывает в переменную file_info информацию о следующем элементе каталога и возвращает значение GNOME_VFS_ERROR_EOF, если прочитана вся директория. Мы считываем информацию о директории из полей file_info. Если одна и та же структура GnomeVFSFileInfo используется в программе несколько раз, перед повторным использованием ее следует очищать с помощью функции gnome_vfs_file_info_clear().

Конечно, в программах GNOME можно обойтись и без GNOME VFS, но использование этой системы позволяет значительно расширить функциональность программы без особых трудозатрат со стороны программиста. Библиотеки GNOME VFS – лишь часть обширного набора библиотек GNOME. Вспомогательным библиотекам GNOME будет посвящена следующая, заключительная статья этой серии. LXF

Через месяц Мы разберемся с наиболее интересными библиотеками GNOME и подключим к работе C++.