

RНР-GTK – вторая попытка

Кирилл Сухов

В день, который впоследствии будет затруднительно отмечать, а именно 29 февраля этого года, после нескольких лет разработки и тестирования, общественности был представлен новый релиз – пакет RНР-GTK2. Данный продукт представляет собой интерфейс RНР для работы с библиотекой GTK2 и предназначен для построения на данном языке, применяющемся преимущественно для веб-программирования, полноценных настольных оконных приложений.

Предыстория

RНР-GTK (<http://gtk.php.net>), пакет для написания на RНР настольных GUI-приложений появился на свет благодаря Андрею Змиевски (Andrei Zmievski), одному из основателей Zend inc. Андрея вдохновило появление пакетов PyGtk и GTK-Perl, и он решил попробовать сделать средства для работы с GTK+ на RНР. Заручившись поддержкой создателя PyGtk (инструмента для работы с Gtk+ на языке Python) Джеймса Хестриджа (James Henstridge), он начал работу. Чуть позже к ней присоединился программист Фрэнки Крованни (Frank'y Kromann'y). Первый релиз RНР-GTK был представлен общественности 1 марта 2003 года. Проект зажил самостоятельной жизнью с собственным сайтом и CVS-репозитарием.

Казалось бы всё было отлично – тысячи разработчиков получили в свои руки кроссплатформенный инструмент для разработки оконных приложений на RНР, но всё оказалось не так просто. Большинство программистов встретили этот продукт не с распростёртыми объятиями. Самый главный вопрос, который возникал, был «Зачем?» В самом деле, зачем нужен подобный инструментарий, когда настольные приложения можно писать на Си + Gtk или на MSVC++? Преимущества использования скриптового языка оценены не были, и широкого распространения технология не получила. Довольно долго о проекте мало что было слышно. Нет, редкие новости на сайте появлялись, было видно, что работа ведётся, но и только. И вот теперь у нас есть RНР-GTK 2. Какие новые возможности он нам даёт? В чём его отличия от предшественника? Сложится ли судьба этой версии удачной? Чтобы попытаться ответить на эти вопросы, предлагаю установить новинку и опробовать её в деле.

Установка

Прежде всего следует обратить внимание на то, что RНР-GTK2 работает с версией RНР-интерпретатора не ниже 5.1. Впрочем это не особенно важно, так как установка в большинстве случаев предполагает собственный экземпляр интерпретатора.

Пакет доступен для платформ Windows, MacOS, Linux и (как сказано в документации) может быть установлен на большинство POSIX-совместимых операционных систем. Сначала попробуем установить этот пакет на операционную систему Linux (был использован дистрибутив OpenSUSE 10.3).

Сразу неважные новости – никаких скомпилированных бинарных пакетов пока недоступно, и ставить пакет придётся по старинке – из исходных текстов. Необходимо выбрать вариант установки – RНР-GTK2 предполагает два способа инсталляции – с использованием уже существующего RНР интерпретатора и отдельную, автономную установку с собственным интерпретатором. Второй способ рекомендован для неопытных пользователей, но мне кажется, он предпочтителен в любом случае, так как не только снимает требования по версиям RНР и GTK+ (2.6 на всякий случай), но и поможет избежать путаницы в конфигурациях. Итак, приступим:

```
$ cvs -d :pserver:cvsread@cvs.php.net:/repository login
$ cvs -d :pserver:cvsread@cvs.php.net:/repository co -L
-r PHP_5_2 php-src
```

Таким образом, мы получили свежий RНР из репозитория. Теперь собираем и устанавливаем интерпретатор:

```
$ cd php-src
$ ./buildconf
$ ./configure --prefix=/opt/php5_2 --disable-cgi
```

Всё получилось? Лично у меня не совсем (не угодил версией flex), но это привычные неприятности, будем надеяться, что скоро всё будет доступно из пакетов, а пока движемся дальше:

```
$ make
$ su
$ make install
$ echo extension=php_gtk2.so >> /opt/php5_2/lib/php.ini
$ /opt/php5_2/bin/php-config --extension-dir | xargs -L
echo 'extension_dir=' >> /opt/php5_2/lib/php.ini
```

Впрочем, строго говоря, мы проделали несколько больше работы, чем требуется, так как интерпретатор php можно было взять готовый, со страницы проекта. Но теперь от ручной работы не отвертеться – собираем сам пакет php-gtk:

```
$ cd ..
$ cvs -d :pserver:cvsread@cvs.php.net:/repository co php-gtk
$ cd php-gtk
$ ./buildconf --with-phpize=/opt/php5_2/bin/phpize
$ ./configure --with-php-config=/opt/php5_2/bin/php-config
$ make
$ make install
```

Вроде всё... Хотя нет, руководство рекомендует создать ссылку на новый интерпретатор следующим образом:

```
$ ln -s /opt/php5_2/bin/php /usr/bin/php-gtk
```

И добавить одну строчку в конфигурационный файл php.ini:

```
extension=php_gtk2.so
```

А теперь небольшая проверка:

```
$ ln -s /opt/php5_2/bin/php /usr/bin/php-gtk
```

```
linux-ns7:/home/geol/Documents # /opt/php5_2/bin/php -m
[PHP Modules]
ctype
date
dom
filter
hash
iconv
json
libxml
pcre
PDO
pdo_sqlite
php-gtk
...
```

Что и требовалось доказать – расширение php-gtk установлено.

Под Windows всё гораздо проще – нужно скачать бинарный пакет, расположенный на <http://gtk.php.net/download.php>, и распаковать zip-архив в нужную директорию. В директорию php-gtk, образовавшуюся после распаковки архива, следует скопировать файлы php.exe (CLI-версия) и php5ts.dll, а также dll-файлы расширений PHP, которые планируется использовать. Теперь в системную переменную PATH, нужно добавить путь до php-gtk, и пакет готов к использованию.

Впрочем, есть ещё более простой путь – использовать менеджер Gnome (<http://gnome.org>). Всё что тут требуется – скачать, запустить инсталлятор и следовать появляющимся инструкциям. После нескольких кликов мышкой мы получим полностью работоспособные PHP 5.1, GTK 2.6 и пакет PEAR.

Хеллоуворд

Теперь попробуем сваять небольшое демонстрационное приложение. Пока не будем отступать от канона и создадим окошко, используя конструкции, несильно выходящие за рамки первой версии пакета.

Сохраняем следующий код в файл test.phpw:

```
<?php
if (!class_exists('gtk')) {
    die("Please load the php-gtk2 module in your \r\n";
    php.ini\r\n");
}

$wnd = new GtkWindow();
$wnd->set_title('hello php-gtk2');

$wnd->set_default_size(200,300);

$pixbuf = $wnd->render_icon(
    Gtk::STOCK_EXECUTE,
    Gtk::ICON_SIZE_DIALOG
);
$wnd->set_icon($pixbuf);
$wnd->connect_simple('destroy', array('gtk', 'main_quit'));
```

```
$lblHello = new GtkLabel("Yes!\r\n'Hello php-gtk2!'");
$wnd->add($lblHello);

$wnd->show_all();
Gtk::main();
?>
```

Теперь запускаем наше приложение командой:

```
$ php test.phpw
```

Если всё прошло гладко, должен быть виден результат, аналогичный показанному на **рис. 1**. Теперь давайте разберёмся, как нам это удалось. Как видите, в первых строчках скрипта осуществляется проверка на существование загруженного модуля php-gtk2, далее командой:

```
$wnd = new GtkWindow();
```

создаётся новый объект – виджет, экземпляр класса окна. Пользователи первой версии php-gtk, наверное, обратят внимание на отсутствие амперсанда перед ключевым словом new. Всё верно – времена объектной модели PHP 4 прошли. Теперь задаём свойства окна:

```
$wnd->set_title('php-gtk2'); //заголовок
$wnd->set_default_size(200,300); //размер
```

Устанавливаем стильную иконку (стандартный набор представлений тулбар-иконок доступен в документации – <http://gtk.php.net/manual/en/gtk.enum.stockitems.php>):

```
$pixbuf = $wnd->render_icon(
    Gtk::STOCK_EXECUTE,
    Gtk::ICON_SIZE_DIALOG
);
$wnd->set_icon($pixbuf);
```

Далее позаботимся о корректном закрытии окна:

```
$wnd->connect_simple('destroy', array('gtk', 'main_quit'));
```

В данном случае мы связываем сигнал destroy с обратным (callback) вызовом статического метода Gtk::main_quit, который будет применен при закрытии/уничтожении окна.

Надо хорошо понимать, что в отличие от обычных PHP-скриптов, по завершении которых, как известно, интерпретатор завершает работу и освобождает память, здесь приложение продолжает выполняться. Для завершения его работы мы использовали специальный сигнал, с которым связали обработчик обратного вызова. Если бы мы этого не сделали, завершить работу приложения штатными средствами было бы нельзя.

Теперь, наверное, стоит остановиться и коротко пояснить, о каких виджетах и сигналах я говорю. Впрочем, если вы уже работали с gtk, эти термины должны быть вам знакомы, в противном случае, поясню:



Рисунок 1.
Здравствуй,
здравствуй...

- **Виджет** – любой компонент графического интерфейса, будь то главное окно программы, окно диалога, поле ввода или кнопка.
- **Контейнер** – любой виджет, способный содержать в себе другие виджеты.
- **Сигнал** – сообщение, создаваемое виджетом. Если вы никогда не сталкивались с механизмом сообщений, то ближайшим, (правда, грубым) аналогом будет механизм событий в JavaScript (onmousemove, onclick и т. д.). Действительно, некоторые сообщения довольно схожи – например при щелчке левой кнопки мыши на область, занимаемую виджетом, им генерируется сигнал «clicked» и т. д. Для обработки сигналов используются функции обратного вызова.

Строчками:

```
$lblHello = new GtkLabel("Yes!\nHello php-gtk2!");
$wnd->add($lblHello);
```

создается и добавляется в окно новый виджет – ярлык (label), с заданным текстом. Пока показом этого текста функционал нашего приложения и будет ограничен, посему осталось только сделать наше приложение видимым:

```
$wnd->show_all();
```

Специально уточню, для пользователей первой версии пакета: в php-gtk2 метод show() тоже присутствует, но он делает видимым только само окно приложения, без (в данном случае) ярлыка. Теперь запускаем приложение:

```
Gtk::main();
```

Пока наше приложение небогато какими-либо возможностями, поэтому следующим этапом предлагаю заставить его что-нибудь делать.

Добавляем функциональность

Для демонстрации работы сигналов отяготим наше приложение работой – пусть при нажатии специальной кнопки в окошке будут показываться дата и время.

В первую очередь добавим в окно программы ещё один виджет – эту самую кнопку:

```
$button = new GtkButton('press me');
$button->set_relief(Gtk::RELIEF_HALF);
```

Методы и свойства нового объекта подробно освещены в руководстве по php-gtk 2 (<http://gtk.php.net/docs.php>), ознакомиться труда не составляет, проблема в другом. При добавлении кнопки к основному окну:

```
$wnd->add($button);
```

скрипт даёт следующее сообщение об ошибке:

```
Gtk-WARNING **: Attempting to add a widget with type GtkButton to
a GtkWindow, but as a GtkBin subclass a GtkWindow can only
contain one widget at a time; it already contains a widget
of type GtkLabel
```

Ничего удивительного, GtkWindow не может иметь бо-

лее одного потомка (при этом, являясь виджетом верхнего уровня, сам этот объект ничьим потомком выступать не может). Несмотря на предупреждение, сценарий будет выполнен, но никакой кнопки в окне приложения мы не увидим. Выход заключается в использовании тегов контейнеров – GtkHBox или GtkVBox (разница между ними в ориентации компоновки):

```
$wnd->set_icon($pixmap);
$wnd->connect_simple('destroy', array('gtk', 'main_quit'));

$box = new GtkVBox();
$box->set_spacing(4);
// Добавляем кнопку
$button = new GtkButton('press me');
$button->set_relief(Gtk::RELIEF_HALF);
// Добавляем надпись
$lbl = new GtkLabel(date('Y-m-d h:i:s'));
// Пакуем
$box->pack_start($button, false, true, 10);
$box->pack_start($lbl, true, true);
// Помещаем бокс в окно
$wnd->add($box);
```

Командами pack_start() и/или pack_end() мы компоуем виджеты в контейнере-боксе, а его, в свою очередь, добавляем в основное окно программы.

Теперь привяжем к кнопке желаемое исполняемое действие. Сначала напишем функцию, принимающую ссылку на объект окно, как параметр:

```
function foo(GtkLabel $lbl){
    $lbl->set_text(date('Y-m-d h:i:s'));
}
```

теперь свяжем сигнал «clicked» кнопки с этой функцией уже знакомым оператором:

```
$button->connect_simple('clicked', 'foo', $lbl);
```

Как видите, третьим параметром является аргумент функции – ссылка на объект типа GtkLabel. Если бы пришлось передавать несколько аргументов, их следовало бы упаковать в массив. Если всё сделано правильно, после запуска скрипта мы можем насладиться результатом. (см. **рис. 2**).

Отличия

Теперь самое время посмотреть, что же изменилось в PHP-GTK по сравнению с первой версией. Разработчики заявляют, что постарались сделать продукт максимально обратно совместимым, но они же заявляют, что удалось это далеко не во всем. Логично: если бы не было такой платы за прогресс, не было бы и прогресса.

К главному отличию в работе с новой версии пакета сам PHP-GTK имеет косвенное отношение. Дело в новой объектной модели, появившейся в пятой версии PHP. Теперь нет необходимости для передачи по ссылке на объект использовать ампер-



Рисунок 2. Создали нечто полезное... ну почти

санд (&) и конструкции, которые в PHP-GTK1/PHP4 приводили к созданию копий, теперь нормально работают с экземплярами класса. В примере выше это уже было продемонстрировано.

Вместе с PHP-GTK1 ушли в прошлое некоторые виджеты и методы. Рассмотрим **таблицу** с эквивалентами устаревших элементов в PHP-GTK 2.

В документации также присутствует список устаревших методов, причём состоит он всего из одного элемента – вместо метода `set_policy()` теперь следует употреблять `set_resizable`. Впрочем, не стоит особенно оболящаться – там же указано, что таких методов (да и виджетов) потенциально может быть больше, просто не всё ещё задокументировано. Надеюсь, это будет исправлено в ближайшее время, а мы двигаемся дальше. Теперь пришло время создать что-нибудь по-настоящему полезное.

Создаем кроссплатформенное приложение за 20 минут

Сейчас мы создадим простую программу-ежедневник. Она должна будет включать календарь, поле для ввода записей, возможность их просмотра и удаления.

При изучении руководства по PHP-GTK 2, можно обнаружить, что все компоненты, необходимые для создания графического пользовательского интерфейса такого приложения, уже присутствуют в пакете. Посему незамедлительно приступим.

Прежде всего создадим главное окно:

```
$window = new GtkWindow();
$window->set_title("GtkHBox and GtkVBox packing demonstration");
$window->set_position(Gtk::WIN_POS_CENTER);
$window->connect_simple("destroy", array("gtk", "main_quit"));
$window->set_default_size(400,400);
$window->show();
```

Завершение программы – запуск жизненного цикла приложения:

```
$window->show_all();
Gtk::main();
```

Строго говоря, пока метод `show_all()` тут не нужен, но сейчас мы обеспечим его работой. Сначала позаботимся о потомке окна (который, как мы помним, может быть только один):

```
$vbox = new GtkVBox(false, 5);
$window->add($vbox);
```

Теперь создадим и поместим в получившийся бокс необходимые нам элементы. Сначала ярлык:

```
$label = new GtkLabel();
$label->set_text("My daily");
$label->set_justify(Gtk::JUSTIFY_LEFT);
$vbox->pack_start($label, true, true, 5);
```

Далее создаём панель с кнопками – тут понадобится ещё один контейнер – объект `GtkHButtonBox()`:

```
$hbbbox = new GtkHButtonBox();
$hbbbox->set_layout(Gtk::BUTTONBOX_SPREAD);
$hbbbox->set_spacing(15);
```

Эквиваленты устаревших элементов в PHP-GTK 2

Устаревший виджет	Замена
GtkCTree	GtkTreeView/GtkTreeStore
GtkCList и GtkList	GtkTreeView/GtkListStore
GtkCombo и GtkOptionMenu	GtkComboBox/GtkTreeModel
GtkItemFactory	GtkUIManager
GtkOldEditable	GtkEditable
GtkPixmap and GtkPreview	GtkImage/GdkPixbuf
GtkText	GtkTextView/GtkTextBuffer

Тут мы создали панель, задали стиль расположения кнопок и расстояние между ними. Теперь сами кнопки:

```
$button1 = new GtkButton('Exit');
$button2 = new GtkButton('Add');
$button3 = new GtkButton('Clean');
```

Затем упаковываем их в `GtkHButtonBox`, а его, в свою очередь, в основной контейнер приложения:

```
$hbbbox->add($button1);
$hbbbox->add($button2);
$hbbbox->add($button3);
$vbox->pack_end($hbbbox);
```

Продолжаем творить GUI. Познакомимся ещё с одним контейнером `GtkHPaned`. Он будет группировать элементы в нижней части наружного бокса:

```
$vpane = new GtkHPaned();
$vpane->set_border_width(5);
$vbox->pack_end($vpane);
```

Теперь создадим Календарь (элемент `GtkCalendar()`) и разместим его в левой части панели:

```
$left = new GtkFrame();
$day=new GtkCalendar();
$left->add($day);
$left->set_shadow_type(Gtk::SHADOW_IN);
$vpane->add1($left);
```

С правой частью панели несколько сложнее. Следует подготовить её содержимое заранее. Сначала создаём текстовый буфер – объект для хранения и редактирования текста:

```
$textBuffer = new GtkTextBuffer();
$textBuffer->set_text(date('Y-m-d h:i:s'));
```

Ещё один объект – `GtkTextView()` необходим для показа текстового буфера:

```
$text = new GtkTextView();
$text->set_buffer($textBuffer);
```

В первой версии нашей программы не будем предоставлять возможность редактировать записи (чтобы уложиться в оговоренные 20 минут):

```
$text->set_editable(false);
```

Теперь поле ввода записи. Тут подойдёт объект `GtkEntry`, представляющий собой однострочное поле ввода:

```
$entry= new GtkEntry();
```

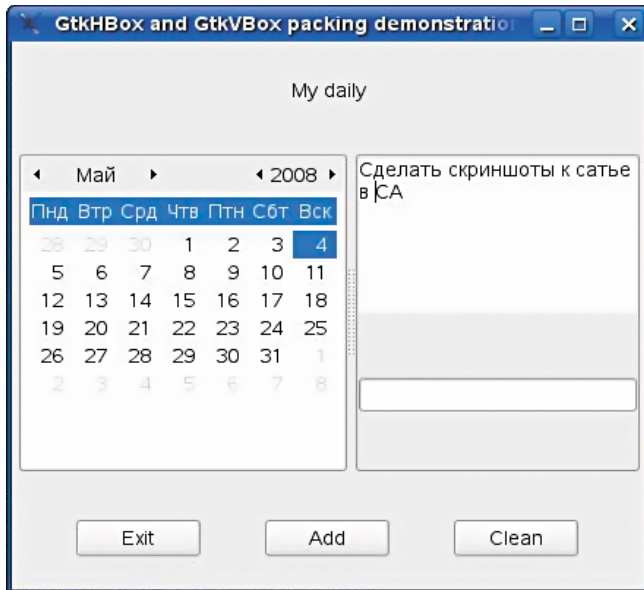


Рисунок 3. Ежедневник. Пока примитивный, но уже вполне рабочий

Чтобы всё скомпоновать в правой части панели, сделаем ещё один контейнер и всё соберём:

```
$r_vbox = new GtkVBox(false, 5);
$r_vbox->pack_end($entry);
$r_vbox->pack_end($text);

$right = new GtkFrame();
$right->add($r_vbox);
$right->set_shadow_type(Gtk::SHADOW_IN);

$vpane->add2($right);
```

С интерфейсом всё. При запуске программы должно появляться окно, изображённое на **рис. 3**.

Теперь определим функции нашего GUI-приложения и свяжем необходимые сигналы виджетов с их реализацией. Итак, при выборе даты на календаре должна открываться запись, соответствующая этой дате (если таковая существует). Связываем сигнал и функцию:

```
$day->connect_simple('day-selected', 'get_day', $day, &
$textBuffer);
```

В данном случае `day-selected` – сигнал объекта `GtkCalendar`, возникающий при выборе текущей даты, объекты `$day` и `$textBuffer` выступают в роли параметров функции `get_day()`, которую мы сейчас и напишем:

```
function get_day(GtkCalendar $d, GtkTextBuffer $bf) {
    $f_name = get_file_name($d);
    if (file_exists('text/' . $f_name)) {
        $content = file_get_contents('text/' . $f_name);
    } else {
        $content = 'No notes';
    }
    $bf->set_text($content);
}
```

Как видите, ничего сложного – мы загружаем в текстовый буфер содержимое файла, а ежели файл отсутствует, пишем про отсутствие записей за этот день. Файлы с записями будем именовать по датам, которым они соответствуют. Напишем для этого одну небольшую функцию:

```
function get_file_name(GtkCalendar $d) {
    $data = $d->get_date();
    return join('-', $data);
}
```

Необходимо отметить, что с функциями можно обращаться как угодно, в рамках привычного PHP-кода, за исключением одного – при использовании объектов в качестве параметров необходимо указывать их тип.

Далее реализуем функцию добавления записи, с которой свяжем сигнал `clicked` кнопки «Add»:

```
$button2->connect_simple('clicked', "add_note", &
$day, $entry, $textBuffer);
```

И тело функции:

```
function add_note(GtkCalendar $d, GtkEntry $entry,
GtkTextBuffer $bf) {
    $f_name = get_file_name($d);
    if (!is_file('text/' . $f_name)) {
        file_put_contents('text/' . $f_name, $entry->get_text());
    } else {
        $fd = fopen('text/' . $f_name, 'a+');
        fwrite($fd, "\n" . $entry->get_text());
        fclose($fd);
    }
    $content = file_get_contents('text/' . $f_name);
    $bf->set_text($content);
}
```

Кнопка «Clean» должна при нажатии уничтожать все записи. Это реализовать еще легче. Связывание:

```
$button3->connect_simple('clicked', "clean_note", $day, &
$textBuffer);
```

и реализация:

```
function clean_note(GtkCalendar $d, GtkTextBuffer $bf) {
    $f_name = get_file_name($d);
    if (is_file('text/' . $f_name)) {
        unlink('text/' . $f_name);
    }
    $bf->set_text('No notes');
}
```

Всё? Нет, мы забыли про кнопку «Exit». Тут просто:

```
$button1->connect_simple('clicked', array("gtk", &
"main_quit"));
```

Вот и всё! Кроссплатформенное, вполне функциональное приложение работает! Правда, там есть ещё над чем потрудиться, это касается как внешнего вида, так и функциональности, но это я предлагаю выполнить самостоятельно. Полный код приложения приведён ниже:

```
<?php

// Создаём директорию для текстовых файлов
if (!is_dir("text")) {
    mkdir("text", 0700);
}

// Снабжаем файл именем, соответствующим дате
function get_file_name(GtkCalendar $d) {
    $data = $d->get_date();
    return join('-', $data);
}
```

```
// Получаем запись из файла и помещаем её на текстовое поле
function get_day(GtkCalendar $d, GtkTextBuffer $bf){
    $f_name=get_file_name($d);
    if(file_exists('text/'.$f_name)){
        $content=file_get_contents('text/'.$f_name);
    }else{
        $content='No notes';
    }
    $bf->set_text($content);
}

// Добавляем запись
function add_note(GtkCalendar $d, GtkEntry $entry, GtkTextBuffer $bf){
    $f_name=get_file_name($d);
    if(!is_file('text/'.$f_name)){
        file_put_contents('text/'.$f_name, $entry->get_text());
    }else{
        $fd=fopen('text/'.$f_name, 'a+');
        fwrite($fd, "\n".$entry->get_text());
        fclose($fd);
    }
    $content=file_get_contents('text/'.$f_name);
    $bf->set_text($content);
}

// Уничтожаем запись
function clean_note(GtkCalendar $d, GtkTextBuffer $bf){
    $f_name=get_file_name($d);
    if(is_file('text/'.$f_name)){
        unlink('text/'.$f_name);
    }
    $bf->set_text('No notes');
}

$window = new GtkWindow();
$window->set_title("GtkHBox and GtkVBox packing demonstration");
$window->set_position(Gtk::WIN_POS_CENTER);
$window->connect_simple("destroy", array("gtk", "main_quit"));
$window->set_default_size(400,400);
$window->show();

$vbox = new GtkVBox(false, 5);
$window->add($vbox);

$label = new GtkLabel();
$label->set_text("My daily");
$label->set_justify(Gtk::JUSTIFY_LEFT);
$vbox->pack_start($label, true, true, 5);
$label->show();

$hbbbox = new GtkHButtonBox();
$hbbbox->set_layout(Gtk::BUTTONBOX_SPREAD);
$hbbbox->set_spacing(15);

$button1 = new GtkButton('Exit');
$button2 = new GtkButton('Add');
$button3 = new GtkButton('Clean');

$hbbbox->add($button1);
$hbbbox->add($button2);
$hbbbox->add($button3);
$vbox->pack_end($hbbbox);

$vpane = new GtkHPaned();
$vpane->set_border_width(5);

$left = new GtkFrame();
$day=new GtkCalendar();
$left->add($day);
$left->set_shadow_type(Gtk::SHADOW_IN);
$vpane->add1($left);

$textBuffer = new GtkTextBuffer();
$textBuffer->set_text(date('Y-m-d h:i:s'));
$text = new GtkTextView();
$text->set_buffer($textBuffer);
```

```
$text->set_editable(true);
$entry= new GtkEntry();
$right = new GtkFrame();
$r_vbox = new GtkVBox(false, 5);
$r_vbox->pack_end($entry);
$r_vbox->pack_end($text);
$right->add($r_vbox);
//$right->add($text);
$right->set_shadow_type(Gtk::SHADOW_IN);
$vpane->add2($right);

$vbox->pack_end($vpane);
$button1->connect_simple('clicked', array("gtk", "main_quit"));
$button2->connect_simple('clicked', "add_note", $day, $entry, $textBuffer);
$button3->connect_simple('clicked', "clean_note", $day, $textBuffer);
$day->connect_simple('day-selected', 'get_day', $day, $textBuffer);

$window->show_all();
Gtk::main();
?>
```

Работа приложения показана на **рис. 3**.

Для иллюстрации использования других, более «продвинутых», виджетов приведу код, ставший впоследствии основой для довольно функциональной системы управления заданиями, применяемой у нас в компании. В данном случае показано дерево групп задач, распределённых по различным отделам IT-департамента. Пояснения даны в комментариях к коду:

```
<?php

// Создаём горизонтальное меню
$mbar = new GtkMenuBar();

// Добавляем пункт горизонтального меню
$file = new GtkMenuItem('_File');

// Добавляем подменю
$fmenu = new GtkMenu();
$file->set_submenu($fmenu);
$mbar->add($file);

$view = new GtkMenuItem('_View');
$vmenu = new GtkMenu();
$view->set_submenu($vmenu);
$mbar->add($view);

$edit = new GtkMenuItem('_Edit');
$emenu = new GtkMenu();
$edit->set_submenu($emenu);
$mbar->add($edit);

$mode = new GtkMenuItem('_Status');
$mmenu = new GtkMenu();
$mode->set_submenu($mmenu);
$mbar->add($mode);

// Теперь создаём пункты подменю для каждого пункта
// в особо тяжких случаях разделяем пункты сепараторами
// меню «File»
$fmenu->add(new GtkImageMenuItem(Gtk::STOCK_NEW));
$fmenu->add(new GtkImageMenuItem(Gtk::STOCK_OPEN));

$fmenu->add(new GtkSeparatorMenuItem());

$fmenu->add(new GtkImageMenuItem(Gtk::STOCK_SAVE));
$fmenu->add(new GtkImageMenuItem(Gtk::STOCK_SAVE_AS));

$fmenu->add(new GtkSeparatorMenuItem());

$fmenu->add(new GtkImageMenuItem(Gtk::STOCK_QUIT));

// Меню вид
$vmenu->add(new GtkImageMenuItem(Gtk::STOCK_SORT_ASCENDING));
$vmenu->add(new GtkImageMenuItem(Gtk::STOCK_ZOOM_FIT));
```

```
// Меню редактирования
$menu->add(new GtkImageMenuItem(Gtk::STOCK_COPY));
$menu->add(new GtkImageMenuItem(Gtk::STOCK_CUT));
$menu->add(new GtkImageMenuItem(Gtk::STOCK_PASTE));

// Меню статуса
$mmenu->add(new GtkImageMenuItem(Gtk::STOCK_APPLY));
$mmenu->add(new GtkImageMenuItem(Gtk::STOCK_CONVERT));

$vbox = new GtkVBox();
$vbox->pack_start($mbar, false);

// Создаём окно
$wnd = new GtkWindow();
$wnd->connect_simple('destroy', array('gtk', 'main_quit'));
$wnd->set_default_size(400, 200);

// Создаём древовидную структуру
$store = new GtkTreeStore(64, 32);

// Создаём «ветви» первого уровня (первый параметр-
// идентификатор родительского элемента. В данном случае
// равен NULL)
$support = $store->append(null, array('Support servise', 1));
$www = $store->append(null, array('Internet', 2));
$c1 = $store->append(null, array('1C', 1));

// Создаём дочерние элементы
$store->append($c1, array('Stuff', 8));
$store->append($www, array('Site', 2));
$store->append($www, array('Ithernet', 12));
$store->append($support, array('Workstantions', 4));

// Создаём отображение дерева
$view = new GtkTreeView($store);
$view->set_enable_tree_lines(enabled);
$cell_renderer = new GtkCellRendererText();

// Добавляем заголовки
$view->append_column(new GtkTreeViewColumn(
    'Department', $cell_renderer, 'text', 0));
$view->append_column(new GtkTreeViewColumn(
    'Problems', $cell_renderer, 'text', 1));

// Разворачиваем дерево
$view->expand_all();
$vbox->pack_start($view, false);

// Помещаем его в окно и делаем видимым
$wnd->add($vbox);
$wnd->show_all();
Gtk::main();
?>
```

Результат – на **рис. 4**.

Богатые графические пользовательские интерфейсы, это, конечно, здорово, но с любым усложнением для их создания приходится писать всё больше однообразного кода. Проблему быстрого проектирования GUI призван решить инструмент, который теперь существенно дополняет PHP-GTK.

Дизайнер пользовательских интерфейсов Glade2

Glade – это приложение для визуального построения графических интерфейсов, на основе GTK+. Оно входит во все популярные дистрибутивы Linux, но при необходимости может быть скачано с официального сайта проекта – <http://glade.gnome.org>. Glade позволяет проектировать графический интерфейс пользователя, просто набрасывая мышкой на будущее окно приложения компоненты из предоставленной палитры (см. **рис. 6**), и сохранять получившееся безобразия в XML-файл в формате GladeXML для последующего использования Gtk+ приложениями, реализованными на различных языках. В PHP-GTK также есть средства работы с GladeXML.

Интерфейс, показанный на **рис. 5**, будет сохранён в виде следующего glade-файла (листинг дан с сокращениями):

```
<?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="window">
  <property name="visible">True</property>
  <property name="title">
    translatable="yes">Glade</property>
  <property name="type">GTK_WINDOW_TOPLEVEL</property>
  <property name="window_position">
    GTK_WIN_POS_NONE</property>
  <property name="modal">False</property>
  <property name="resizable">True</property>
  <property name="destroy_with_parent">False</property>
  <property name="decorated">True</property>
  <property name="skip_taskbar_hint">False</property>
  <property name="skip_pager_hint">False</property>
  <property name="type_hint">
    GDK_WINDOW_TYPE_HINT_NORMAL</property>
  <property name="gravity">
    GDK_GRAVITY_NORTH_WEST</property>
  <property name="focus_on_map">True</property>
  <property name="urgency_hint">False</property>

  <child>
    <widget class="GtkVBox" id="vbox1">
      <property name="visible">True</property>
      <property name="homogeneous">False</property>
      <property name="spacing">0</property>

      <child>
        <widget class="GtkNotebook" id="notebook1">
          <property name="visible">True</property>
          <property name="can_focus">True</property>
          <property name="show_tabs">True</property>
          <property name="show_border">True</property>
          <property name="tab_pos">GTK_POS_TOP</property>
          <property name="scrollable">False</property>
          <property name="enable_popup">False</property>

          <child>
            <widget class="GtkLabel" id="label4">
              <property name="visible">True</property>
              <property name="label" translatable="yes">
                Hellow Glade!</property>
              <property name="use_underline">
                False</property>
              <property name="use_markup">False</property>
              <property name="justify">
                GTK_JUSTIFY_LEFT</property>
              <property name="wrap">False</property>
              <property name="selectable">False</property>
              <property name="xalign">0.5</property>
              <property name="yalign">0.5</property>
              <property name="xpad">0</property>
              <property name="ypad">0</property>
              <property name="ellipsize">
                PANGO_ELLIPSIZE_NONE</property>
              <property name="width_chars">-1</property>
              <property name="single_line_mode">
                False</property>
              <property name="angle">0</property>
            </widget>
          </child>
          <property name="tab_expand">False</property>
          <property name="tab_fill">True</property>
        </packing>
      </child>

      <child>
        <widget class="GtkLabel" id="label1">
          <property name="visible">True</property>
          .....
          <property name="single_line_mode">
            False</property>
          <property name="angle">0</property>
        </widget>
      </child>
    </packing>
  </child>
  <property name="type">tab</property>
</packing>
</child>
```

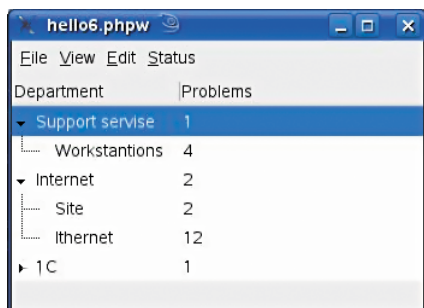


Рисунок 4. Рабочее приложение

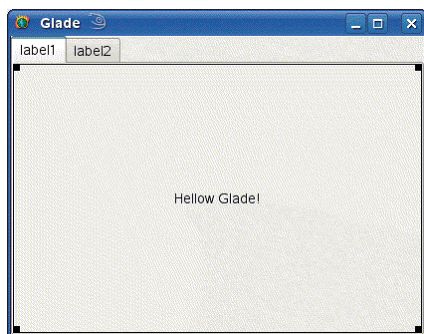


Рисунок 5. Допроектировались

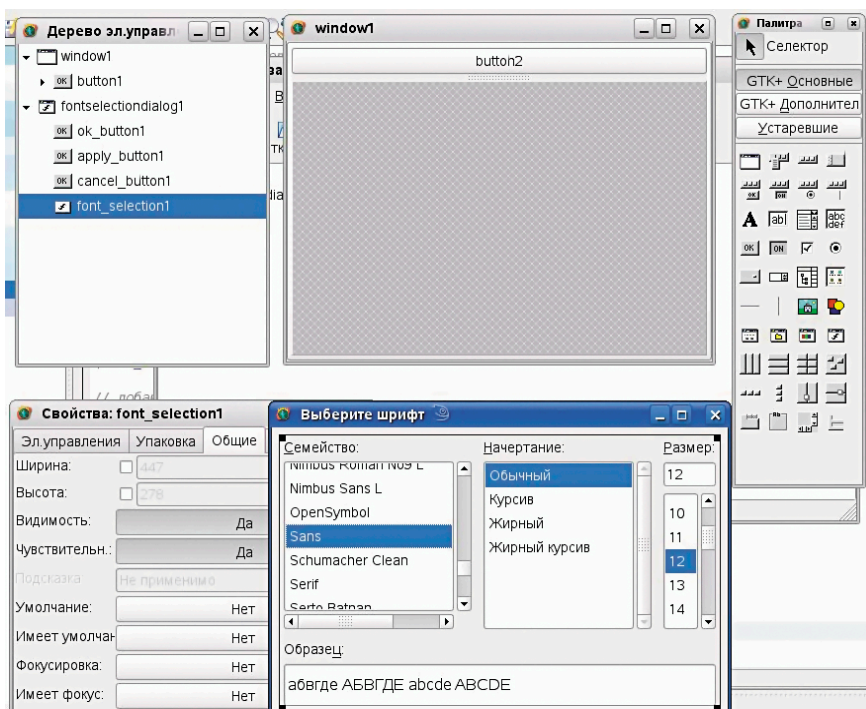


Рисунок 6. Проектируем графический интерфейс с glade

```
<child>
  <placeholder/>
</child>

<child>
  <widget class="GtkLabel" id="label2">
    .....
  </widget>
  <packing>
    <property name="type">tab</property>
  </packing>
</child>
</widget>
<packing>
  <property name="padding">0</property>
  <property name="expand">True</property>
  <property name="fill">True</property>
</packing>
</child>
</widget>
</child>
</widget>
</glade-interface>
```

Использовать такой файл в качестве готового интерфейса можно следующим образом:

```
<?php
$glade = new GladeXML('testglade.glade');
Gtk::main();
?>
```

Это, разумеется, самый простой случай, но и в связывании сигналов объектов и событий нет ничего сложного:

```
<?php
$glade = new GladeXML('testglade.glade');
$window = $glade->get_widget('wndClose');
$window->connect_simple('destroy', array('Gtk', 'main_quit'));

$button = $glade->get_widget('btnClose');
$button->connect_simple('clicked', 'onClickButton');

function onClickButton() {
    echo "button clicked!\n";
}
```


```
Gtk::main_quit();
}Gtk::main();
?>
```

Как видите, с таким инструментарием создание функционального настольного приложения на PHP не только возможно, но и до безобразия легко осуществимо!

Заключение

PHP-GTK 2, кроме поддержки объектной модели PHP5, Glade и GTK2+, принёс с собой много новых виджетов, объектов, методов. Подробно об их работе лучше прочитать в документации проекта, которая, может, пока и недостаточно полна, но постоянно улучшается (кстати, в этом процессе можно принять участие). Целью же моего опуса было привлечь внимание к новому этапу данной технологии. Получит ли PHP-GTK широкое распространение? Есть ли вообще будущее у PHP в области настольных приложений?

Да, конечно, возможности пакета по сравнению с первой версией значительно возросли, но будут ли они востребованы? Кинуться ли тысячи PHP-кодеров писать клиентские приложения? Мне кажется, такого в массовом не случится, так как ниша настольных приложений давно и прочно занята более продвинутыми конкурентами.

В то же время определённые продвижения в этом направлении вполне возможны. PHP, конечно, не потеснит на этом рынке Java или C, но PHP-GTK-приложения вполне имеют право на жизнь. Нас ведь давно не удивляют GUI-программы написанные на Python? 

1. Официальный сайт проекта PHP-GTK – <http://gtk.php.net>.
2. Уваров А. PHP-GTK. //Системный администратор, №12, 2004 г. – С. 60-61 (<http://www.samag.ru/cgi-bin/go.pl?q=articles;n=12.2004;a=13>).
3. Документация на PHP-GTK 2 – <http://gtk.php.net/manual/en>.
4. Сайт проекта Glade – <http://glade.gnome.org>.