Потрошим Gimp

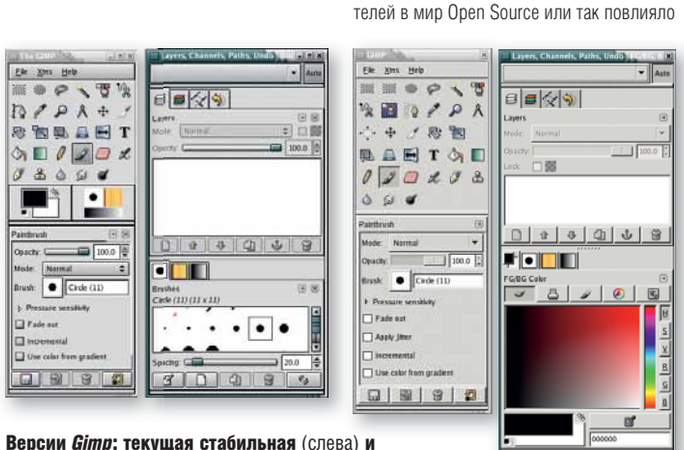
Есть много способов участвовать в разработке открытого проекта – например, помогать в написании документации или непосредственно создавать код. Почему бы не начать с Gimp? **Майкл Дж. Хэммел** проведет вас через весь процесс исправления ошибок, который он припас заранее...

НА ДИСКЕ

- ***Gimp 2.2.11* и *2.3.8***
- **Снимок *GECL CVS***

***GNU Image Manipulation Program* – более известный как *GIMP* – дедушка настольных приложений в мире открытых программ. Он вступил в** жизнь как *Motif*-приложение в 1995 г. и привел к созданию *Gimp Toolkit* (также известного как GTK) и рабочего стола Gnome.

Хотя существуют настольные приложения и постарше, чем *Gimp*, ни одно из них не привлекло столько новых пользователей в мир Open Source или так повлияло на настольные системы. Но даже и проекты-дедушки нуждаются в постоянной поддержке открытого сообщества: разработчиков, писателей, художников, журналистов и пользователей.



Версии *Gimp*: текущая стабильная (слева) **и разрабатываемая** (справа)**. Обе имеются на диске, и вы можете их исследовать.**

Как помочь

Разработка *Gimp* – как и многих других открытых проектов – целиком зависит от информации от пользователей. Списки пожеланий и запросы на возможности посылаются через Bugzilla – это web-система, предназначенная для отслеживания статуса ошибок – и просматриваются командой разработчиков *Gimp*. Принятые пожелания, а также набор требуемых исправлений, перебираются в грядущий релиз. Пока разработчики вгрызаются в новые возможности, ошибки и изменения документации, пользователи тестируют выпущенные версии.

В конечном счете новый стабильный релиз становится доступным в виде исходных текстов. Поставщики Linux, такие, как Red Hat, Mandriva и Ubuntu, затем собирают новые версии и включают в свои дистрибутивы.

Разработка осуществляется двумя очень маленькими командами: основной командой и рядом участников. Основная команда фактически состоит всего из двух человек: Свена Ньюмена [Sven Neumann] и Митча Неттерера [Mitch Natterer]. *Ньюмен де факто* является лидером проекта, однако проект структурирован не жестко, поэтому важные решения часто принимаются только после серьезного изучения откликов остальных членов команды.

Активные участники – группа человек в 30, они работают над различными разделами проекта, включая исходный код, документацию и управление ресурсами типа репозитария *CVS*. Другие разработчики прошлого (включая автора этой статьи!) размещены на вкладке окна About.

Для помощи проекту вам не обязательно уметь кодировать, но вы должны быть хорошо знакомы с приложением *Gimp* с точки зрения конечного пользователя.

Ваша миссия

В этой статье мы продемонстрируем два лучших способа помочь проекту: охоту за ошибками и их исправление.

Охота за ошибками может быть случайной или целевой. Целевое тестирование ставит своей задачей изолировать проблему и описать ее более подробно –

Элита команды *Gimp*: ведущие разработчики Митч Неттерер (третий слева внизу) и Свен Ньюман (четвертый слева в среднем ряду).

РЕСУРСЫ УЧАСТНИКА

- **http://developer.gimp.org**
Информация для разработчиков *Gimp*
- **www.gimp.org/bugs/howtos/bugzilla.html** Описание ошибок
- **www.gimp.org/bugs/howtos/submit-patch.html** О заплатках
- **www.le-hacker.org/papers/gobject**
Объектная система *GLib*
- **http://developer.gimp.org/api/2.0/index.html** Документация по *API Gimp*
- О стилях написания кода см. файл **HACKING** в исходном коде.

Все это обязательно к прочтению. Однако если вы хотите реального взаимодействия с другими участниками, то надо подключиться к почтовым рассылкам и IRC-каналам:

- **www.gimp.org/mail_lists.html**
- Канал **#gimp** на **irc.gimp.org**

Существует много почтовых рассылок проекта. Основными являются рассылка для пользователей *Gimp* (по основным вопросам использования) и для разработчиков *Gimp* (по основным вопросам разработки). Живое обсуждение происходит на IRC-каналах. Более подробную информацию см. в wiki на **http://wiki.gimp.org/gimp/irc**.

это особенно актуально для трудновоспроизводимых ошибок. Охотники за ошибками должны быть очень хорошо знакомы со стабильной версией *Gimp*, а целенаправленные охотники – уметь компилировать новые версии программы. Познакомьтесь также с Bugzilla, мы представим ее вам на стр. 50.

Что касается исправления ошибок, то ядро *Gimp* написано на C, но есть и много дополнительных модулей на различных языках скриптов: самые популярные – Perl, Python и Script-Fu (вариация Scheme). Вы должны быть знакомы с созданием заплаток; инструкции можно найти на странице **www.Gimp.org**, но мы рассмотрим этот процесс более подробно на стр. 52.

Еще до начала...

Если вы собираетесь порыться в исходном коде, то прежде вам необходимо знать о нескольких инструментах.

CVS

Каждый участник *Gimp* знаком с *CVS*, программным обеспечением, которое управляет исходным кодом *Gimp*. Пользователи получают код из *CVS*, производят изменения, создают заплатку и отправляют ее команде разработчиков или участникам.

Bugzilla

Bugzilla – web-система, используемая для отслеживания изменений в проекте *Gimp*. Это могут быть сообщения об ошибках, запросы на новые возможности или изменения в документации. С первого раза Bugzilla немного ошеломляет, но команда *Gimp* усердно поработала, чтобы облегчить вхождение в свой мир. На странице *Gimp* вы найдете всю информацию об использовании Bugzilla для сообщений об ошибках и их поиске; а на странице разработчиков *Gimp* (*см. врезку Ресурсы участника*) содержатся более подробные указания для опытных пользователей Bugzilla, о том как отыскать уже имеющиеся сообщения.

C, Autoconf и другие инструменты

Исходный код ядра *Gimp* использует набор инструментов Autoconf (*Autoconf, Automake, Libtool*) для управления сборкой. Знание Autoconf желательно, но не обязательно. Зато важно умение программировать на C – оно требуется для работы с ядром *Gimp* и многими модулями. Для некоторых модулей необходимо знать Perl, Python и Script-Fu, но для работы с кодом ядра эти языки не потребуются. Пригодится знакомство с текстовым редактором и такими инструментами, как *Cscope*, *Strace*, *GDB*. Надо также вникнуть в стиль программирования *Gimp* (*снова, см. врезку Ресурсы участника*).

Не программист?

Прежде чем начать охоту за ошибками, кратко опишем другие способы помочь проекту *Gimp*.

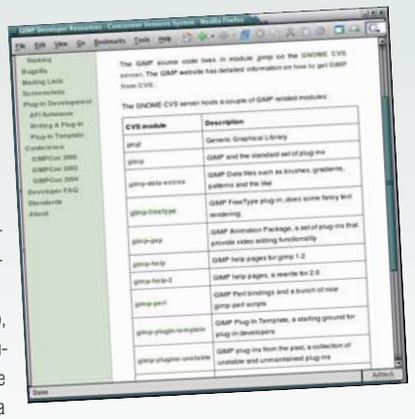
Первый – поддержка пользователей. Команда разработчиков проводит большую часть своего свободного времени за работой над исходным кодом, поэтому у них мало времени на ответы новичкам. Достаточно освоившись с программой, вы можете обеспечивать поддержку конечных пользователей посредством списков рассылок, web-страниц и чатов.

Такая помощь очень важна: ваша поддержка – видимая часть проекта, обращенная к публике. Поэтому обязательно надо полностью понимать, как пользоваться программой, и вежливо реагировать даже на самую жесткую критику.

Также очень ценны переводы и документация. В переводе нуждается как документация, так и интерфейс программы. Это прекрасная возможность помочь проекту: работа над большей частью документации требует лишь знания какого-либо текстового редактора, а работа над web-сайтами – знания XML и формата DocBook.

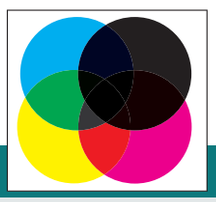
Опытные программисты могут попробовать задуматься о расширении ключевых возможностей. Общеизвестный пример – поддержка 16-битовых цветовых каналов. Потребность в ней назрела давно, однако ее реализация потребовала неожиданно большого объема работ. В итоге запланирован пересмотр исходного кода *Gimp*: он будет использовать код вспомогательного проекта под названием GEGL (*см. врезку внизу*).

В какой форме ни выразится ваше участие, *Gimp* представляет собой отличный старт, и мы надеемся, что опыт этой захватывающей работы послужит вам хорошей наградой. Прежде чем нырнуть в исходный код, взгляните на врезку Ресурсы участника, расположенную вверху. Если мы вдохновили вас на поиск и исправление ошибок, можете начинать – просто переверните страничку!



***Gimp* содержит много проектов, под управлением системы контроля исходного кода *CVS*.**

>>



GEGL: БУДУЩЕЕ GIMP

GEGL означает *Generic Graphical Library*; де факто это будущее проекта *Gimp*. Пока проект находится в процессе разработки, а когда будет закончен, станет внутренней библиотекой обработки в *Gimp*. Кроме поддержки 16-битовых цветовых каналов (одна из первоначальных целей), *GEGL* предоставит богатую функциональность для обработки изображений, включая управление цветом, редактирование цветовых пространств CMYK (см. картинку вверху справа) и L*a*b, гибкие системы на основе мозаики и многопоточную обработку изображений. Также он предоставит механизм на основе направленного графа для запоминания серии манипуляций над изображением.

Большинство этих возможностей требуется профессиональным пользователям – видеоредакторам, например. Но и средний пользователь также найдет их полезными. Так, механизм на основе направленного графа позволяет хранить сложную последовательность шагов при создании изображения, от добавления текста до применения фильтров, и даже редактировать эти шаги. Допустим, вы изготовили в *Gimp* логотип для web-странички и сохранили ваши действия в файле. Позже вы можете загрузить файл и создать, например, логотип для рекламного плаката – указав другое разрешение или размер холста.

Официальной даты вхождения *GEGL* в состав *Gimp* не существует, но есть надежда увидеть его в очередной главной версии после *Gimp 2.4*. А до тех пор проект нуждается в разработчиках и тестировщиках – и вы знаете, что делать.

www.gegl.org

<< Часть 1 Выходим на охоту

Прежде чем исправлять ошибки, надо сначала о них узнать – поэтому заглянем в Bugzilla!

Лучший способ понять, что такое работа над проектом – это исправить в нем ошибку.

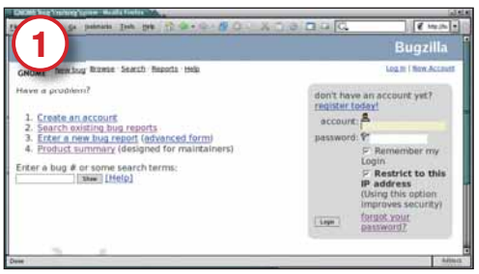
С точки зрения вклада в проект *Gimp*, ошибкой считается все, что существует в исходном коде *Gimp* и работает не так, как должно – в отличие от запроса на возможность, для которой исходного кода либо нет вообще, либо он есть, но спроектирован нежелательным образом.

Если ошибка обнаружена, очень важно суметь ее снова воспроизвести, тут-то и пригодится ваша помощь: просто играйте с программой, пока не сведете к минимуму число факторов, вызывающих ошибку. Затем опишите их в виде последовательности шагов, чтобы любой желающий смог прочесть ваше сообщение в Bugzilla.

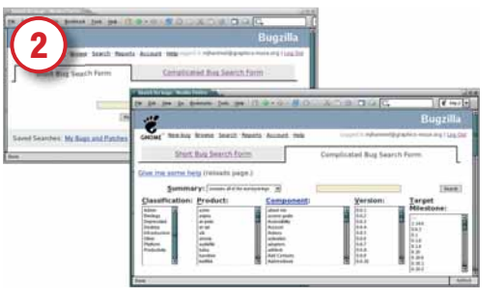
База Bugzilla (**http://bugzilla.gnome.org**) служит для многих проектов, связанных с рабочей средой Gnome. Прежде чем искать в ней сообщения об ошибках, вам понадобится зарегистрироваться и получить ID.

Выбрав ошибку по вкусу, скачайте пос-

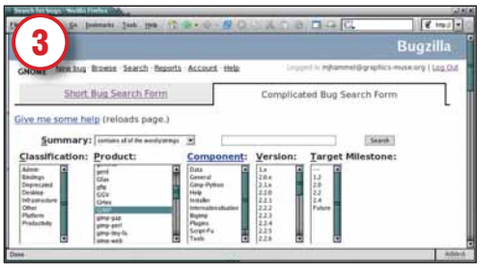
КАК НАЙТИ СООБЩЕНИЯ ОБ ОШИБКАХ В BUGZILLA



Войдите в базу данных Bugzilla и нажмите на ссылку Search вверху страницы, затем перейдите к Complicated Bug Search Form – Запросу на расширенный поиск. Для поиска ошибок воспользуемся специальным ключевым словом gnome-love, им команда разработчиков *Gimp* помечает легко устранимые ошибки, чтобы их можно было легко найти.



При поиске сообщений об ошибках надо обратить внимание на два момента. Первый –список продуктов. Листайте его до тех пор, пока не наткнетесь на запись Gimp. Нажмите на нее. Произойдет обновление других списков, но мы пока их проигнорируем.



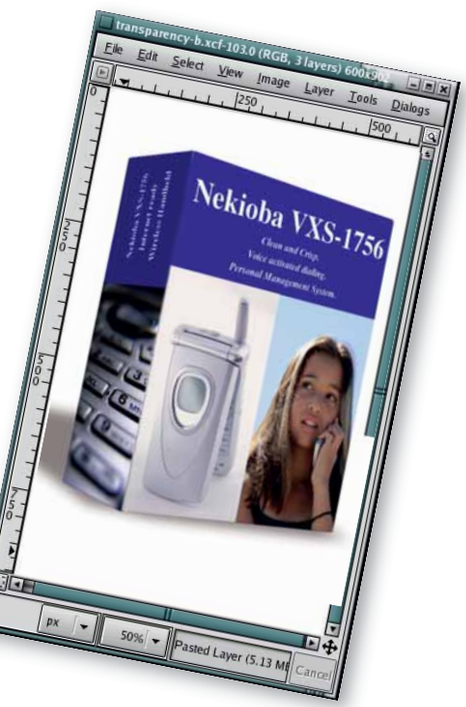
Прокрутите экран до области, означенной как Advanced Searching Using Boolean Charts – Продвинутый Булев поиск. Там есть два выпадающих списка. В первом списке выберите Keywords (Ключевые слова), а во втором – Contains The String (Содержит строку). В текстовом поле рядом со списком наберите gnome-love. Поиск подготовлен.

Нажмите кнопку Search (она находится прямо над списками, которые вы только что модифицировали). Просмотрите колонку Summary – если какая-либо ошибка вам приглянется, нажмите на ее номер, чтобы узнать о ней больше (*см. таблицу ниже*). Мы выбрали ошибку #331839.

5

ID	Sev	Pri	OS	Product	Status	Resolution	Summary
331839	enh	Nor	All	Gimp	NEW		Clear keyboard shortcuts

Данная таблица информирует о выбранной ошибке Gimp. Во втором столбце (Степень серьезности ошибки) стоит enh, сокращение от enhancement (улучшение), т.е. формально это не ошибка, а запрос на улучшение. (Таблицу заполняют разработчики, получив через Bugzilla извещение об ошибке.) В данном случае решено изменить диалог Preferences: добавить возможность очистить клавиши быстрого доступа без их сброса в первоначальное состояние. Мы думаем, что сможем с этим справиться. Вот и попробуем.



леднюю копию *Gimp* из *CVS*. Чтобы она заработала, понадобятся следующие зависимости:

- *autoconf 2.54* или старше.
- *automake 1.7* или старше.
- *libtool 1.4* или старше.
- *gettext 0.13* или старше.
- *GTK 2.8.10* или старше.

Возможно, первые четыре пакета у вас есть, но бьемся об заклад, что последнего нету. *GTK* имеет свои зависимости. Установить их легко, но надо позаботиться, чтобы они ничего не сломали в вашей системе.

Итак, скачайте и распакуйте *GTK*. В любом современном дистрибутиве вам скорее всего понадобятся последние *GTK*, *Glib*, *ATK*, *Pango* и Cairo. Первые четыре доступны на сайте *GTK* (**www.gtk.org/download**). Архив Cairo также содержится на сайте *GTK*, но на другой странице (**ftp://ftp.gtkK.org/pub/gtk/v2.8/dependencies**). Важно собрать пакеты в правильном порядке: Cairo, *GLib*, *Pango*, *ATK*, *GTK*.

Перед сборкой скажем каждому пакету, где искать библиотеки и информацию о конфигурации:

```
export PKG_CONFIG_PATH=/usr/
    local/gtk+-2.8/lib/pkgconfig:$PKG_
    CONFIG_PATH
export LD_LIBRARY_PATH=/usr/
    local/gtk+-2.8/lib:$LD_LIBRARY_
    PATH
```

Установите эти переменные среды, иначе в сборку попадут старые версии, уже установленные на вашей системе. В директории каждого пакета наберите команды:

```
./configure --prefix=/usr/local/gtk+-
    2.8
make
sudo make install
```

Мы тем самым указываем всем пакетам устанавливаться в **/usr/local/GTK+-2.8**. Благодаря этому новые версии ПО не будут мешать старым.

Сборка

Установив зависимости *GTK*, соберите *Gimp*. Необходимо зайти на *CVS*-сервер и скачать исходный код. Направьте вашу машину на *CVS*-сервер *Gimp*:

```
export CVSROOT=':pserver:
    anonymous@anonCVS.Gimp.org:/cvs/
    gnome'
```

и зайдите:

```
cvs login
```

Вы получите примерно следующее сообщение:

```
'Logging in to :pserver:
anonymous@anonCVS.Gimp.
org:2401/CVS/gnome
CVS password:'
```

Мы заходим как анонимный пользователь, поэтому у нас будут права только на чтение. Просто нажмите Enter (пароль вводить не надо).

Теперь скачайте исходный код *Gimp*:

```
cvs -z3 checkout Gimp
```

Флаг **-z3** говорит об использовании сжатия, для ускорения передачи файлов [и экономии трафика, – прим.ред.].

Готово? В вашем текущем каталоге должен появиться каталог **gimp**. Чтобы собрать код, наберите:

```
cd Gimp
./autogen.sh --prefix=/usr/local/
Gimp-2.3
make
sudo make install
```

Скрипт **autogen.sh** похож на скрипт *configure*, который вы запускали для *GTK*-приложений, и теперь вы передаете опцию **--prefix** скрипту **autogen.sh**, а не configure (заметим, что мы ставим *Gimp* в директорию, отличную от *GTK*). Переменные **PKG_CONFIG_PATH** и **LD_LIBRARY_PATH**, установленные перед сборкой *GTK*, укажут системе сборки *Gimp*, где искать *GTK*. Мы хотим поставить *Gimp* в отдельный каталог, чтобы потом мы смогли его удалить и собрать другую версию, не повредив *GTK*.

План атаки

Теперь мы можем осмотреть самую последнюю версию диалога Preferences и составить план, как исправить наш досадный недочет. Интересующая нас область обведена и выделена синим цветом на рисунке (*справа вверху*) – это опции Keyboard Shortcuts. В сообщении говорилось, что требуется опция очистки горячих клавиш и что существующие кнопки необходимо изменить на меню опций, подобное меню Navigation Preview Size.

Очень хорошо... Поищем код, ответственный за создание этой части диалога Preferences. В исходном коде *Gimp* находится много каталогов, включая:

- **app** код ядра *Gimp*.
- **data** Кисти, градиенты и прочее.
- **plugins** Фильтры и другие дополнительные модули.
- **po** Переводы текстов в интерфейсе пользователя, например, в меню.

Нам требуется через каталог **app** добраться до каталога **dialogs**, а в нем найти файл **preferences-dialog.c**. В файле поищите строку, которая находится на кнопке Reset,. Таким образом мы найдем код, с помощью которого создается эта кнопка:

```
button2 = prefs_button_add (
Gimp_STOCK_RESET,
_("_Reset Saved Keyboard Shortcuts
    to "
"Default Values"),
GTK_BOX (vbox2));
```

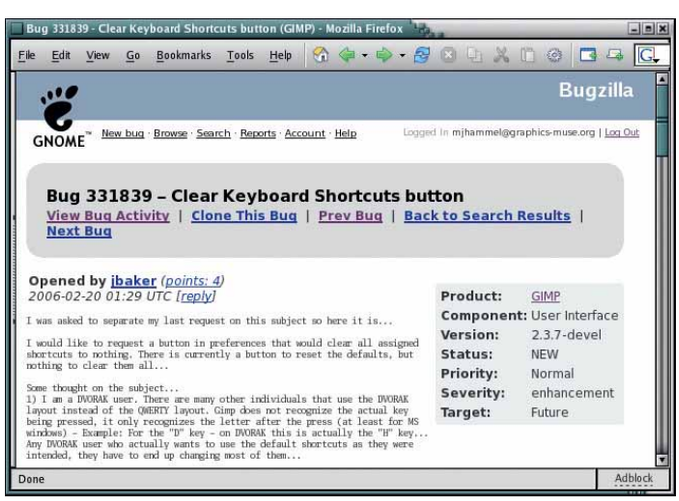
Вот что мы хотим изменить: кнопки, выделенные синим цветом, надо переделать на список опций. Думаете, справимся? Конечно, справимся!

```
g_signal_connect (
button2, "clicked",
G_CALLBACK (prefs_menus_clear_
    callback),
Gimp);
g_object_set_data (
G_OBJECT (button),
"clear-button", button2);
```

Ага, вызовы **prefs_button_add()** надо заменить на один вызов функции **gimp_int_combo_box_new()**, создающей виджет меню – ее можно найти в документации по API *Gimp* (**http://developer.Gimp.org/api/2.0/libGimpwidgets/GimpIntComboBox.html**). Кроме того, обратные вызовы для обеих кнопок надо слить в один обратный вызов.

Перечень мероприятий готов; приступим же к его выполнению.

>>



Сообщение об ошибке может послать любой – только опишите ее поподробнее.

<< Часть 2 Поправки и заплатки

Задача поставлена – давайте выполним ее и отошлем результат команде разработчиков на проверку....

После трудов по локализации ошибки исправление оказывается простейшей частью. Мы описали его здесь, и когда вы будете исправлять другой код, шаги, которые мы предпримем, вам помогут.

Исправление

Добавим поле со списком прямо под существующими кнопками в **app/dialogs/preferences-dialog.c**. Чтобы пояснить, зачем это поле нужно, надпишем его (слева). Все это требует *GTK*-виджета 'hbox':

```
hbox = gtk_hbox_new (FALSE, 6);
gtk_box_pack_start (GTK_BOX
    (vbox2), hbox, FALSE, FALSE, 0);
gtk_widget_show (hbox);
label = gtk_label_new (_("Keyboard
    Shortcut Status:"));
gtk_box_pack_start (GTK_BOX
    (hbox), label, FALSE, FALSE, 0);
gtk_widget_show (label);
```

Затем добавим поле со списком из четырех записей. Первая запись – «пустышка»: она позволит сбросить меню после каждого действия, чтобы пользователь в любой момент смог перезапустить то же самое действие.

```
combo = Gimp_int_combo_box_new
    (
_("Choose an Action"),
    KBS_IGNORE,
_("Save Now"),
    KBS_SAVE,
_("Reset to Default Values"),
    KBS_RESET,
_("Remove All Keyboard Shortcuts"),
    KBS_CLEAR,
    NULL);
Gimp_int_combo_box_set_active
    (Gimp_INT_COMBO_BOX (combo),
    KBS_SAVE);
gtk_box_pack_start (GTK_BOX
(hbox), combo, FALSE, FALSE, 0);
    gtk_widget_show (combo);
g_signal_connect (combo,
    "changed",
G_CALLBACK (prefs_menus_
    keyboard_shortcuts),
Gimp)
```

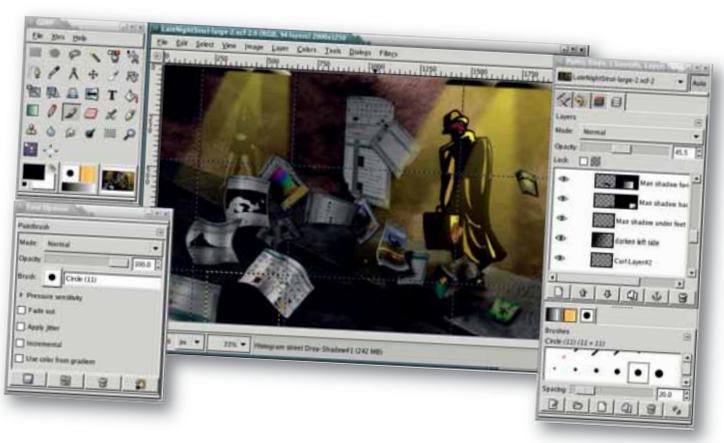
Строка **g_signal_connect()** сообщает программе, что когда пользователь изменяет наше новое поле со списком, надо вызвать функцию **prefs_menus_keyboard_shortcuts()**. Эта функция получает ID виджета поля со списком (ID используется для обнаружения, какой пункт меню был выбран) и структуру **gimp**.

Каждый пункт меню в вызове **gimp_int_combo_box_new()** описан рядом с определяемым значением (**KBS_SAVE** и т.д.). Объединим их в перечисляемый тип и добавим его в начале файла:

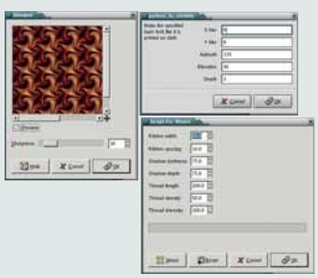
```
enum {
KBS_IGNORE,
KBS_SAVE,
KBS_RESET,
KBS_CLEAR
};
```

Теперь надо заменить обратные вызовы функций для существующих кнопок одной функцией. Любые изменения в поле со списком должны осуществлять функции обратного вызова. Это просто. Новая функция будет просматривать поле со списком, определять, какой пункт меню был выбран, и использовать оператор **switch** для выбора соответствующего действия. В двух случаях (**KBS_SAVE** и **KBS_CLEAR**) действие останется как у прежних кнопок. Переключаясь на первый, пустой пункт меню, не будем делать ничего. Последний пункт «чистит» горячие клавиши: проходится по списку и сбрасывает их. После этого новый список требуется сохранить – и он будет доступен в следующий раз, когда пользователь запустит программу.

```
static void
prefs_menus_keyboard_
shortcuts(GtkWidget *combo,
Gimp *Gimp)
{
gint value;
gint fd;
GError *error = NULL;
gchar *filename;
Gimp_int_combo_box_get_active
(Gimp_INT_COMBO_BOX (combo),
&value);
switch(value)
```



ПИШЕМ ДОПОЛНИТЕЛЬНЫЙ МОДУЛЬ К GIMP



Модули на C, Python и Script-Fu: окна предпросмотра доступны только для написанных на C.

Дополнительные модули – это небольшие программки, написанные на каком-либо языке программирования и расширяющие возможности *Gimp*. Их часто называют фильтрами, просто потому, что большинство из них находится в меню Filters. Однако модуль может вставить себя в любой пункт меню и осуществлять не только обработку изображений, но также генерировать логотипы, шаблоны, обеспечивать ввод-вывод и даже действовать как сервер для внешних программ, чтобы отображать изображения через *Gimp*.

Модуль можно написать на C с использованием библиотеки *libgimp*, или же на одном из поддерживаемых скриптовых языков: Script-Fu (подмножество языка Scheme) и Python. Perl тоже поддерживается, но только как добавочный пакет. Каждый из скриптовых языков связан с API libgimp.

Чтобы написать модуль, нужно быть знакомым с PDB, Procedural Database [База Процедур], содержащей набор всех функций, которые можно вызывать из модуля. Просмотр PDB через осуществляется браузером Procedure Browser (Xtns > Procedure Browser) из меню в Toolbox. Каждую функцию можно искать по имени, причем предоставляется краткое описание параметров. Способ вызова функций зависит от языка, на котором написан модуль. В C вы можете сделать следующее:

```
image_id = gimp_image_
new(width, height, type);
```

тогда как в Perl это может выглядеть так:

```
image_id = $image->new(width,
height, type);
```

а на Python вот так:

```
image_id = gimp.image(width,
height, type)
```

Хороших руководств по написанию модулей немало. Для написания на C смотрите серию руководств из трех частей на сайте разработки *Gimp*, по адресу **http://developer.gimp.org/plug-ins.html**. Отличное руководство для языка Python доступно в сети на **www.gimp.org/docs/python/index.html**.

Документация для Perl входит в состав пакета **Gimp-Perl** – ее можно прочесть, набрав perldoc **Gimp** или посетив страницу **www.gimp.org/tutorials/Basic_Perl**. Желаем удачи!

Наша обновленная панель Keyboard Shortcuts в диалоге Preferences; новое меню отображается в правой нижней части окна.

```
{
case KBS_IGNORE:
break;
case KBS_SAVE:
menus_save (Gimp, TRUE);
g_message (_("Your keyboard
shortcuts have been saved."));
break;
case KBS_RESET:
menus_clear (Gimp, &error);
g_message (_("Your keyboard
shortcuts will be reset to default "
"values the next time you start
Gimp."));
break;
case KBS_CLEAR:
menus_remove (Gimp);
menus_save (Gimp, TRUE);
g_message (_("Your keyboard
shortcuts will be cleared "
"the next time you start Gimp."));
break;
}
Gimp_int_combo_box_set_active
(Gimp_INT_COMBO_BOX (combo),
KBS_IGNORE);
}
```

Мы добавили две новых записи в файл **app/menus/menus.c** и добавили функцию **menus_remove()**, вызываемую из **prefs_menus_keyboard_shortcuts()**. Она производит сброс всех горячих клавиш в цикле.

```
void
menus_remove (Gimp *Gimp)
{
gtk_accel_map_foreach(0,menus_
remove_actions);
}
static void
menus_remove_actions (gpointer
data,
const gchar *accel_path,
guint accel_key,
GdkModifierType accel_mods,
gboolean changed)
{
gtk_accel_map_change_entry
(accel_path, 0, 0, TRUE);
}
```

Нужно также написать прототип функции **menus_remove()** в заголовочном файле **app/menus/menus.h**, чтобы ее можно было вызывать из **preferences-dialog.c**.

Проводим тестирование

Тяжелая работа позади. Выполним перекомпиляцию и переустановку – теперь можно запустить приложение и проверить работу изменений.

```
make
sudo make install
/usr/local/Gimp-2.3/bin/Gimp-2.3
```

Простейший способ проверить изменения – просмотреть в меню File. Прежде чем запускать новую опцию Remove All Keyboard Shortcuts, обратите внимание, что к пунктам меню File и Save приписаны горячие клавиши. Когда мы запускаем новую опцию, эти клавиши исчезают. Выйдем из программы и запустим ее снова – увидим, что клавиш нет. Если затем в меню выбрать пункт Reset, выйти и перезапустить *Gimp*, все горячие клавиши будут восстановлены по умолчанию.

Создание заплатки

Изменения, значит, работают. Время создать заплатку и послать ее на суд почтенным разработчикам *Gimp* – так вы будете поступать и при самостоятельной работе над исправлениями. Из корня каталога исходного дерева *Gimp* наберите

```
cvs diff -up > ../patchfile.patch
```

В каталоге, расположенном на один уровень выше, чем текущий, в котором вы работаете, создастся файл заплатки. Команда *diff* может проработать довольно долго. Проверьте, что файл содержит только изменения, сделанные вами: команда способна протащить туда и постороннюю информацию.

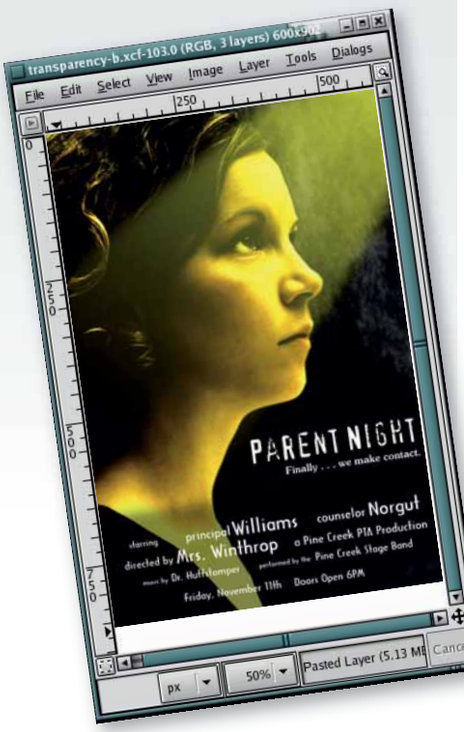
Внизу сообщения об ошибке в Bugzilla есть поле, содержащее информацию о проделанной работе. Заполните секцию Комментарий и присоедините заплатку с помощью ссылки, расположенной ниже. Задача выполнена: можете самодовольно откинуться в кресле.

И последнее. Разработчики ответили, что наша реализация будет работать лучше, если вместо меню будет задействована третья кнопка. Так что мы опять взялись за дело и написали код для третьей кнопки. После этого Митч Неттерер заплатку принял.

Теперь вы поняли: процесс абсолютно прямолинейный, поэтому изучайте, участвуйте и создавайте! LXF

БЛАГОДАРНОСТИ

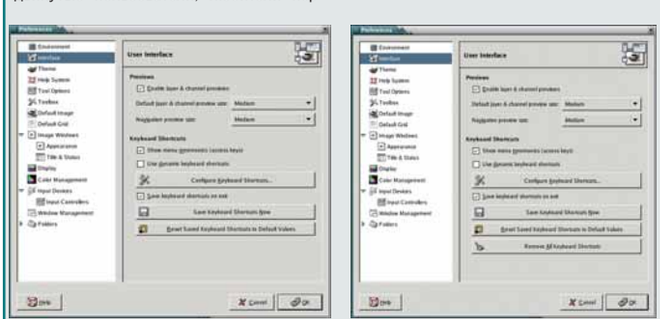
Эта статья основана на слайдах, представленных Карин Делвер [Karine Delvare] на конференции Libre Graphics, прошедшей ранее в этом году. Дополнительную помощь оказали Кэрол Спирс [Carol Spears], Митч Неттерер [Mitch Natterer] и Уильям Скэггс [William Skaggs]. Особая благодарность Карин Делвер (и ее мужу) и Кэрол Спирс за предоставленные фотографии разработчиков на встрече Libre Graphics.



КАКОВА СУДЬБА ИСПРАВЛЕНИЯ LXF?

В нашем исправлении кнопки располагались друг рядом с другом. Но из-за ограничений на ширину текста разработчики решили добавить еще одну кнопку пониже исходных. Обычному пользователю это может не пригодиться никогда, но продвинутые пользователи, любители персональных конфигураций, оценят новую возможность.

В будущем можно добавить варианты предустановок горячих клавиш: выбор конфигурации по умолчанию, как в *Photoshop* или как в Mac OS.



До (слева) **и после** (справа)**: в диалог Preferences добавилась новая опция. Продвинутым пользователям она очень понравится.**