Учебное руководство для новичков

(версия 2016/06/02)

https://github.com/MaxPerl/perl-Gtk3-Tutorial

перевод Кувшинов Д.А. aka vilfred from http://rulinux.net/ (форк http://www.linux.org.ru/), пожелания и примечания по переводу писать сюда: vilfred1@gmail.com

При помощи данного руководства вы сможете ознакомиться с основами графического программирования (GUI) при помощи библиотеки GTK+ с использованием Perl.

Если Вы никогда не программировали прежде или не знакомы с понятием объектно-ориентированного программирования, вам, возможно, стоит прочитать базовые основы этого подхода в книгах "Perl in 21 Tagen"(нем.), "Изучение Perl"(англ.) или любого другого учебного руководства по Perl. Как только вы это сделаете, можно будет вернуться к изучению данного введения

Запуск примеров

Для запуска примеров надо сделать следующее:

1. Введите или скопируйте код примера в файл, затем сохраните с именем подобным filename.pl
2. Для запуска программы введите следующую команду в терминале:

```
perl filename.pl
```

После запуска программы вы увидите либо простейший элемент управления графического интерфейса пользователя, имеющий стандартный внешний вид и выполняющий стандартные действия (англ. widget), либо (если ошиблись при вводе программы) сообщение об ошибке в консоли (или ином элементе управления, обеспечивающем взаимодействие человека с компьютером), которое поможет определить причину проблемы.

Иерархия элементов управления

Повествование этого описания GTK+ Perl будет вестись от простого к сложному, но никто не мешает вам использовать программу , минуя несколько шагов.

Оглавление

2. Рисунки и заголовки

2.1. Рисунки

2.2. Строки

2.3. Заголовки

3. Введение в свойства окна программы

3.1. Свойства и конфигурация состояния базового окна программы

4. Сетка, разделитель и прокрутка

4.1. Сетка

4.2. Разделитель

4.3. Прокрутка в окне (ScrolledWindow)

4.4. Выравнивание (Paned)

5. Сигналы, вызовы и кнопки

5.1. Сигналы и вызовы

5.2. Кнопки

5.3. Гиперссылки

5.4. Чекбоксы

5.5. Кнопки изменения состояния(ToggleButton)

5.6. Переключаетли

5.7. Радио кнопки

6. Группы кнопок(ButtonBox)

6.1. Группы кнопок и их свойства(ButtonBox)

7. Другие виджеты отображения

7.1. Поле состояния(Statusbar)

7.2. Виджет, отображающий работу какого либо процесса(Spinner)

7.3. Виджет, отображающий время работы какого либо процесса(ProgressBar)

8. Виджеты ввода

8.1. Виджет инкремента/декремента(SpinButton)

8.2. Поле текстового ввода(Entry)

8.3. Масштабная линейка(Scale)

9. Виджет для записи и отображения текстаA widget to write and display text

9.1. Текстовое поле(TextView)

10. Диалоговые кнопки и действия

10.1. Диалог

10.2. Пример отображения AboutDialog

10.3. Диалог на примере вывода в консоль(MessageDialog)

11. Меню, панели инструментов и подсказки (использование Glade и GtkBuilder)

11.1. Меню

11.2. Кнопка вызова меню

11.3. Панель инструментов

11.4. Подсказки **(to do)**

11.5. Панель инструментов с помощью Glade

11.6. Кнопка вызова меню при помощи XML и GtkBuilder

12. Кнопки выбора(Selectors)

12.1. Выбор цветовой палитры(ColorButton ***(to do))***

12.2. Выбор шрифта(FontChooserWidget ***(to do))***

12.3. Выбор файлов(FileChooserDialog)

13. Дерево каталогов и меню выбора (TreeViews and ComboBoxes (using the M/V/C design))

13.1. Меню выбора(ComboBox (one column))

13.2. Простое дерево каталогов первого уровня (Simple TreeView with ListStore)

13.3. Дерево каталогов с ветвлением (TreeView with TreeStore)

13.4. The Model/View/Controller design ***(to do)***

13.5. Выкидное меню (ComboBox (two columns))

13.6. Пример дерева каталогов с описанием(More Complex Treeview with ListStore)

13.7. Пример дерева каталогов с ветвлением(More Complex TreeView with TreeStore)

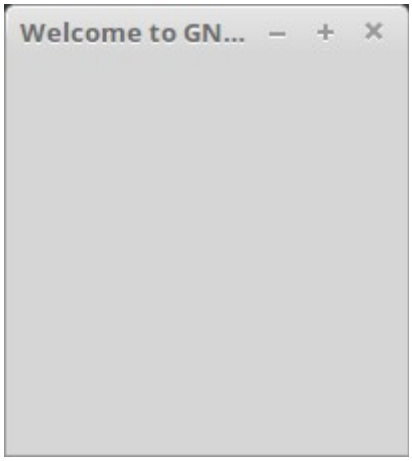
14. Произвольный виджет

14.1. Виджет

1. Основные элементы окна программы

1. 1. Базовое окно

Внешний вид:



Эта программа GtkApplication выглядит как картинка из текущей директории

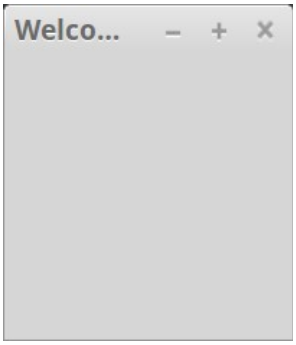
Код, который выводит это окно

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

# создать Gtk Window
my $window = Gtk3::Window->new('toplevel');

# Установить заголовок set the title
$window->set_title('Welcome to GNOME')

# выйти из программы, если пользователь закроет программу
$window->signal_connect('delete_event'=>sub{Gtk3->main_quit()});

# показать окно
$window->show_all;

# создать и запустить приложение
Gtk3->main();
```

Основные свойства для виджета базового окна

- *set_default_size(200, 100)* устанавливает размеры по умолчаниювысотой 200 и шириной 100 пикселов; Если вместо положительного значения передать отрицательное – будут установки по умолчанию.
- *set_position("center")* позиционирование окна. Иметт следующие параметры: *"none"*, *"mouse"*, *"center-always"*, *"center-on-parent"*.

1. 2. Некоторые свойства базового окна[1]



Простейшее GtkApplication окно которое поддерживает Gmenu.

NOTE: Структура GtkApplication приложения отличается от типовой структуры "Gtk3 mainloop" программы. Вы не можете вызвать Gtk3::init – это происходит автоматически. Все инициализация происходит при вызове программы и активирует вызовы(????????). Вместо вызова Gtk3::main и Gtk3::quit, вы вызываете методы GApplication 'run' и 'quit'.

Поэтому нужно импортировать вызов **Glib::IO**, для привязкик Gio API. Существуют два способа:

1) Можно сделать привязку к Glib::IO API в ручную в нашей Perl программе при помощи приведенного кода, помещенного в начало скрипта:

```
BEGIN {
use Glib::Object::Introspection;
Glib::Object::Introspection->setup(
basename => 'Gio',
version => '2.0',
package => 'Glib::IO');
}
```

Это способ, который мы будем стараться использовать в данном руководстве.

1 This chapter was made possible alone by Jeremy Volkening who explained me creating a Gtk3::ApplicationWindow. He has also written the code of this example! Thank you very much for that, Jeremy Volkening!

2) Альтернативный метод можно найти в ранней реализации модуля https://git.gnome.org/browse/perl-Glib-IO (не находится на CPAN!). После загрузки вы можете скопировать *lib/Glib/IO.pm* в *directory_of_your script/Glib/IO.pm* или установить модль как системный с помощью следующих команд:

perl ./Makefile.PL
make
make install

Далее подгрузить модуль в программу обычным образом

use Glib::IO;

Надеемся, что использование Gio API в будущем (для примера преобразование байтов при чтении или записи с помощью Gio API [см. пример в программе FileChooserDialog], не будет излишне в следующих версиях модуля).

Программа, илюстрирующая этот пример

```
#!/usr/bin/perl

# Make a binding to the Gio API in the Perl program (just copy&paste ;-))
# This is necessary mainly for Gtk3::Application
# Alternatively you find an early implementation as a Perl module
# on https://git.gnome.org/browse/perl-Glib-IO (not yet published on CPAN!)
# Hopefully this module simplifies the use of the Gio API in the future
# (see also the notes above).
BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}

use strict;
use warnings;

use Gtk3;
use Glib qw/TRUE FALSE/;

# The MAIN FUNCTION should be as small as possible and do almost nothing except creating
# your Gtk3::Application and running it
# The "real work" should always be done in response to the signals fired by Gtk3::Application.
# see below
my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );
```

```
$app->run(\@ARGV);

exit;


# The CALLBACK FUNCTIONS to the SIGNALS fired by the main function.
# Here we do the "real work" (see above)
sub _init {
	my ($app) = @_;

	# Handle program initialization
	print "Hello world!\n";

 }

sub _build_ui {

	my ($app) = @_;

	my $window = Gtk3::ApplicationWindow->new($app);
	$window->set_title ('Welcome to GNOME');
	$window->set_default_size (200, 200);
	$window->signal_connect( 'delete_event' => sub {$app->quit()} );
	$window->show();

}

sub _cleanup {

	my ($app) = @_;

	# Handle cleanup
	print "Goodbye world!\n";

}
```

Основные свойства для Gtk3::ApplicationWindows

- *set_default_size(200, 100)* устанавливает размеры по умолчаниювысотой 200 и шириной 100 пикселов; Если вместо положительного значения передать отрицательное – будут установки по умолчанию.
- *set_position("center")* позиционирование окна. Имеет следующие параметры: *"none"*, *"mouse"*, *"center-always"*, *"center-on-parent"*.

Для более детального понимания как создаются и запускаются приложения Gtk3::Applications настоятельно рекомендуется прочитать https://wiki.gnome.org/HowDoI/GtkApplication !

Так как данное руководство базируется на аналогичном, но для Python – всегда используется Gtk3:: . Такой подход будет использоваться , в основном, с приложениями, имеющими меню или если это действительно необходимо

Причины этого решения:

1) практическая: Jeremy Volkening убедил меня создать Gtk3::ApplicationWindow после написания данного руководства.

2) Трудности с написанием при примененииотличного от CPAN модуля Glib::IO

3) Я думаю, что привязка к Glib::IO API и создание отличного от стандартного класса Gtk3::ApplicationWindow делают простые примеры очень усложненными и особенности специфических элементов запуска библиотек могут быть объяснены лучше "main-loop"-Style

Если вам проще перевести "mainloop"-style приложение in a Gtk3::Application , то воспользуйтесь следующими шагами:

1) Включите привязку к Gio API в программу как было описано выше.

2) Создайте класс Gtk3::ApplicationWindows для запуска и показа виджета Gtk3::ApplicationWindows и его содержимого как будет показано ниже. Здесь вы должны иметь ввиду смысл сноски. Также отметьте, что вызов *"$app->quit"* вместо *"Gtk3→main_quit" и вызов* Gtk3 необходимо делать без аргумента -init !

```
package MyWindow;
use strict;
use warnings;
use Gtk3;
use Glib qw/TRUE FALSE/;
# Наш класс должен быть подклассомGtk3::ApplicationWindow и
# наследовать методы, свойства и т.п. of Gtk3::ApplicationWindow
use base 'Gtk3::ApplicationWindow' ;

sub new {
        my ($window, $app) = @_;
        $window = bless Gtk3::ApplicationWindow->new($app);
        $window->set_title ("MenuButton Example");
        $window->set_default_size(600,400);
        $window->signal_connect( "delete_event" => sub {$app->quit()} );
#  тут использовать методы, как описано в специальной части
[...]
}

#  также и определить вызовы функций
sub on_click_cb { [...] }
```

3) Создайте приложение Gtk3::Application и запустите его в вашем основном классе. При этом не вызывайте Gtk3::init и используйте *"$app->run;"* вместо *"Gtk3->main();"*. В целом основная часть выглядит так:

```
package main;
```

```
use strict;
use warnings;

# не использовать  -init!!!
use Gtk3;
use Glib qw/TRUE FALSE/;

# создать и запустить the Gtk3::Application
my $app = Gtk3::Application->new('app.id', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;

# работа с сигналами основной функции.
sub _init {
        my ($app) = @_;

        # инициализация дескриптора
        print "Hello world!\n";
}

sub _build_ui {
        my ($app) = @_;
        my $window = MyWindow->new($app);
        $window->show_all();
}

sub _cleanup {
        my ($app) = @_;

        # его уничтожение
        print "Goodbye world!\n";
}
```

2. Рисунки и заголовки

2. 1. Рисунки

Данная программа показывает изображение из директории.

Есил изображение не подгрузилось, будет показана битая картинка в виде иконки "broken image". Файл filename.png .должен находиться в текущей директории

Программный код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;


# создаем базовое окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title('Welcome to GNOME');
$window->set_default_size(300,300);
$window->signal_connect('delete_event'=>sub{Gtk3->main_quit()});

# создаем рисунок
my $image = Gtk3::Image->new();
```

```
# читаем файл filename.png
$image->set_from_file('gnome-image.png');

# добавляем картинку на основное окно
$window->add($image);

# отображаем окно и запускаем приложение
$window->show_all;
Gtk3->main();
```

Основные методы виджета Image

- Для загрузки картинки из сети используется функция *set_from_pixbuf(pixbuf)*, где pixbuf объект GdkPixbuf:

```
#!/usr/bin/perl
use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

# создаем окно
[… см. выше]

# создаем метод pixbuf из файла с именем='gnome-image.png', с шириной=32
# и высотой=64 и болевой переменной preserve_aspect_ratio=False.
my $pixbuf = Gtk3::Gdk::Pixbuf->new_from_file_at_scale('gnome-image.png', 64, 128,
FALSE);

# создаем объект
my $image = Gtk3::Image->new();

# читаем файл filename.png
$image->set_from_pixbuf($pixbuf);

# помещаем картинку на основное окно
$window->add($image);

[… см. ниже!]
```

Если *preserve_aspect_ratio=TRUE* можно использовать *new_from_file_at_size(filename, width, height)*. Если *width* or *height* = -1, программа сама определит размеры.

Для потокового отображения изображений смотрите документацию по функциям *new_from_stream()* и *new_from_stream_at_scale()*.

2. 2. Строки

[***python specific – to do!]***

GTK+ использует UTF-8 для текста. Если умляуты (ä, ö, ü, ß etc) отображаются плохо,

используйте прагму "*use utf8;" в начале скрипта*. Если не помогает, то, видимо, надо декодировать текст, который должен будет отображаться в utf8 кодировкупри помощи *Encode::decode('utf-8', $string)* или, что еще лучше, открыть файл с указанием кодировки '<:encoding(UTF-8)'!

Если вы хотите что-то отображать в терминале вы можете поставить вывод в UTF 8 моду:

```
# set the pragma utf8 to prevent that umlauts of in the script created strings are displayed wrongly
use utf8;
# set the "Line Discipline" of the standard output into the UTF 8 Mode.  Thereby the terminal
# doesn't try to convert the string again to Latin-1
binmode STDOUT, ':utf8';
```

2. 4. Заголовки

Пример заголовка

Код для этого примера

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

# создать окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title('Welcome to GNOME');
$window->set_default_size(200,100);
$window->signal_connect('delete_event'=>sub{Gtk3->main_quit()});

# создать заголовок
my $label = Gtk3::Label->new();

# задать текст заголовка
$label -> set_text('Hello GNOME!');

# поместить виджет заголовка на окно
$window->add($label);

# показать его и запустить все
$window->show_all;
Gtk3->main();
```

Используемые методы для виджета заголовка

Объяснения работы со строками в GTK+ можнго будет найти ниже в главе "Строки".

- *set_line_wrap(TRUE)* - разрешает делать перевод каретки в случае, если длинна текста больше длинны виджета.
- *set_justify("left")* (or "*right*", "*center*", "*fill*")) очуществляет выравнивание текста по границам виджета. Не работает в случае одной строки.
- Для украшения текста *set_markup('text')*, где опция *'text'* – текст в стиле Pango Markup Language. Пример:[2]

```
$label->set_markup   ('Text can be <small>small</small>, <big>big</big>,'.
                      '<b>bold</b>, <i>italic</i> and even point to somewhere'.
                      'in the <a href=\"http::://www.gtk.org\" '.
                      'title=\"Click to find out more\">internets</a>');
```

- для перевода каретки надо выставить перееменную "TRUE" в функцию

```
$label->set_line_wrap(TRUE);
```

3. Свойства виджетов

Введение

Свойства описывают конфигурацию и состояние виджетов. У каждого виджета есть свой собственный определенный набор свойств. Например, у виджета, такого как кнопка есть свойство "метка", которая содержит текст виджета. Вы можете определить имя и значение любого числа свойств при помощи метода, связанного с ним. Например, чтобы создать метку с текстом "Привет Мир”, угол 25 градусов, и выровненной вправо, вы напишете:

```
My $label = Gtk3::Label-new();
$label->set_label('Hello World');
$label->set_angle(25);
$label->set_halign(Gtk.Align.END);
```

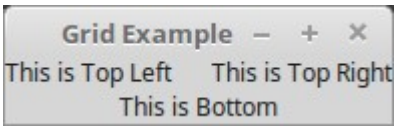
Как только Вы создали такую метку, Вы можете получить текст метки $label->get_label () и аналогично для других свойств.

Также можно получить и установить свойста виджета при помощи функций get_property ('prop_name') и set_property ('prop_name', value), соответственно.

2 Note: At line-breaks inside quotation marks the lines are wraped in the label, too. To prohibit this while at the same time keeping legibility inside the code, the quotation marks are closed and connected to the operator ".".

4. Сетка, разделитель и прокрутка

4. 1. Сетка



Заголовки на сетке

Код для этого примера

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my $window = Gtk3::Window->new('toplevel');
$window->set_title('Grid Example');
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

# несколько заголовков
my $label_top_left = Gtk3::Label->new('This is Top Left');
my $label_top_right = Gtk3::Label->new('This is Top Right');
my $label_bottom = Gtk3::Label->new('This is Bottom');

# секта
my $grid = Gtk3::Grid->new();

# несколько пробелов между колонками сетки
$grid->set_column_spacing(20);

# позиционированние виджета первого заголовка в сетке в левый угол окна:
$grid->attach($label_top_left,0,0,1,1);

# второй заголовок
$grid->attach($label_top_right,1,0,1,1);

# прикрепляем третий заголовок ниже первого заголовка
$grid->attach_next_to($label_bottom, $label_top_left, 'bottom', 2, 1);

# добавляем сетку на панель
$window->add($grid);

# показываем все
$window->show_all;
Gtk3->main();
```

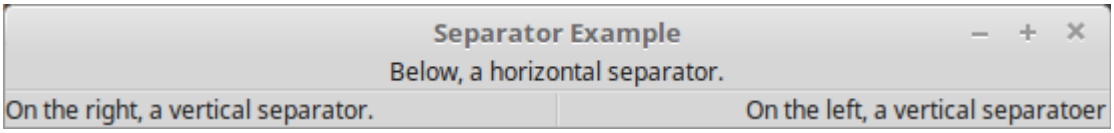
Методы секти

- Для добавления одного дочернего виджета заголовка(label) на виджет родительский,

сетки, виджета на позицию *left*, *top* по координатам *width*, *height* используется функция *attach*(*child*, *left*, top, *width*, *height*). Если на виджете более верхнего уровня(parent) уже находится виджет более низкого уровня, дочерний (child) то при добавлении еще одного виджета, мы можем также использовать функцию *attach_next_to(child, sibling, side, width, height)*, где параметр side принимает следующие значениия: *"left"*, *"right"*, *"top"*, *"bottom"*.

- *Функции insert_row(position)* и *insert_column(position)* сделают то, что им будет сказано относительно позиционированния виджетов в колонки и столбцы, до или после или вниз на строку с соответствующим выравниванием или переносом дочернего виджета на новую строку. Функция *insert_next_to(sibling, side)* добавляет строку или стобец в определенном программистом месте. Новая строка или столбец добавляются в зависимости от положения виджета, если положение *"top"* или *"bottom"*, - вставляется строка, если положение *"left"* или *"right"*, вставляется столбец.
- *Функции set_row_homogeneous(TRUE)* и *set_column_homogeneous(TRUE)* устанавливают, что (соответственно) у каждой строки или каждого столбца есть одинаковая ширина или высота.
- *Функции set_row_spacing(spacing)* и *set_column_spacing(spacing)* принудительно устанавливают интервал между (соответственно) строками или столбцами. Значение интервала может быть между 0, который является значением по умолчанию, и 32767.

4. 2. Разделитель



Горизонтальный и вертинальный разделитель между заголовками(Labels, т.е. строки с текстом).

Соответствующий код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my $window = Gtk3::Window->new('toplevel');
$window->set_title ('Separator Example');
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# три заголовка
my $label1 = Gtk3::Label->new();
$label1->set_text('Below, a horizontal separator.');
```

```
my $label2 = Gtk3::Label->new();
$label2->set_text('On the right, a vertical separator.');

my $label3 = Gtk3::Label->new();
$label3->set_text('On the left, a vertical separatoer');

# горизонтальный разделитель
my $hseparator = Gtk3::Separator->new('horizontal');

# вертикальный разделитель
my $vseparator = Gtk3::Separator->new('vertical');

# добавление виджетов разделителей и заголовков на виджет сетки
my $grid = Gtk3::Grid->new();
$grid -> attach ($label1, 0, 0, 3, 1);
$grid -> attach ($hseparator, 0, 1, 3, 1);
$grid -> attach ($label2, 0, 2, 1, 1);
$grid -> attach ($vseparator, 1, 2, 1, 1);
$grid -> attach ($label3, 2, 2, 1, 1);
$grid -> set_column_homogeneous(TRUE);


$window -> add($grid);


$window -> show_all();
Gtk3->main();
```

4. 3 Виджеты прокрутки

Картинка с виджетами прокрутки.

Соответствующий код

```
#!/usr/bin/perl

use strict;
```

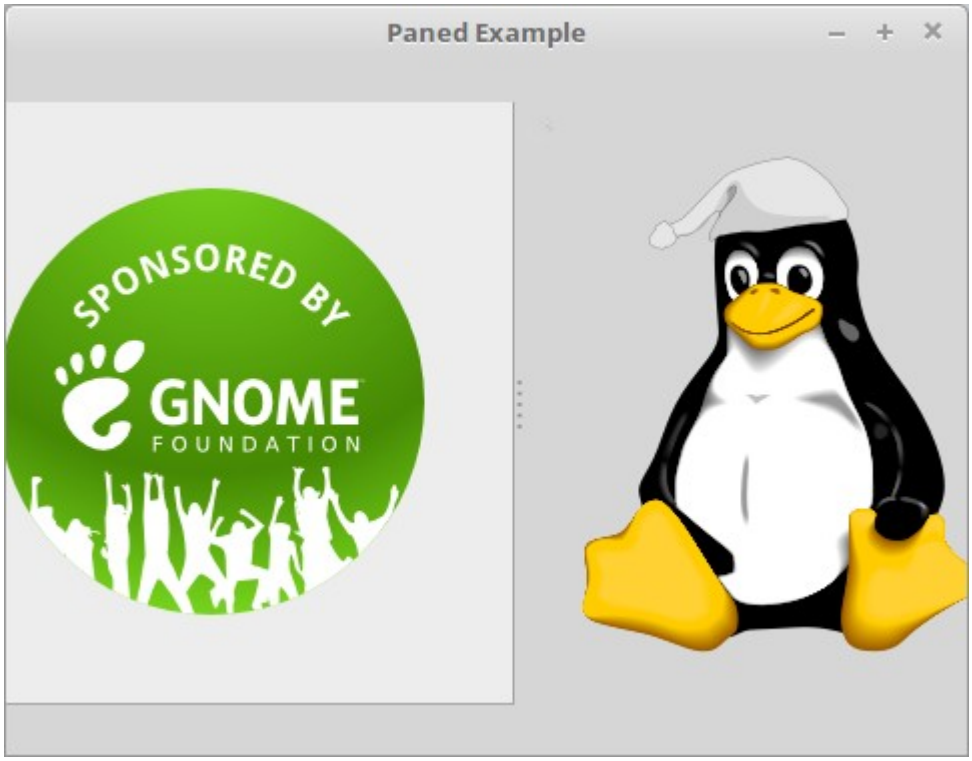
```
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my $window=Gtk3::Window->new('toplevel');
$window->set_title('Scrolled Window Example');
$window->set_default_size(200,200);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# окно с элементами управления прокрутки
my $scrolled_window=Gtk3::ScrolledWindow->new();
$scrolled_window->set_border_width(10);

# установить прокрутку всегда (в противном случае: automatic – если нет – или никогда
$scrolled_window->set_policy('always', 'always');

# изображение - немного больше, чем окно...
my $image = Gtk3::Image->new();
$image->set_from_file('gnome-image.png');

# ставим картинку
$scrolled_window->add_with_viewport($image);

# добавляем полоски прокрутки
$window->add($scrolled_window);

# показываем
$window->show_all();
Gtk3->main;
```

Функции для интерфейса прокрутки (ScrolledWindow widget)

- *set_policy(hscrollbar_policy, vscrollbar_policy)* – каждый из аргументов можетиметь одно из трех значений *"automatic"*, *"always"*, *"never"* которые регулируют появление горизонтальной или вертикальной прокрутки в случае надобности: значение *"automatic" вызывает по мере надобности*, *"always"* и*"never" выставляет прокрутку по принуждению*.
- *add_with_viewport($widget)* указывает Gtk3::Widget $widget отключить прокрутки по умолчанию.
- *set_placement(window_placement)* позиционирует размещение содержания виджета относительно полос прокрутки для прокрученного окна. аргументы are "*top-left"* (по умолчанию прокрутки внизу и справа окна), *"top-right"*, *"bottom-left"*, *"bottom-right"*.
- *set_hadjustment($adjustment)* и *set_vadjustment($adjustment)* устанавливают выравнивание Gtk3::Adjustment $*adjustment*. Это автоопределение(или заданное принудительно)значения верхней и нижней границы вместе с шагом и прибавлением значения числа страниц текста или картинки в виджете и определяется так: *"my $adjustment=Gtk3::Adjustment->new(initial value, minimum value, maximum value, step_increment, page_increment, page_size)"*, поля имет тип *float*. (заметьте что,

параметр step_increment не используется в смысле автоопределения и может быть установлен в 0 0.)

4. 4 Выравнивание(Paned)



Два изображения в двух корректируемых областях, горизонтально выровненных по горизонтали.

Код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my $window = Gtk3::Window->new('toplevel');
$window->set_title ('Paned Example');
$window->set_default_size(450, 350);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# виджет выравнивания налево и направо
my $paned = Gtk3::Paned->new('horizontal');

# две картинки
my $image1 = Gtk3::Image->new();
$image1 -> set_from_file('gnome-image.png');

my $image2 = Gtk3::Image->new();
```

```
$image2 -> set_from_file('tux.png');

# добавляем первую картинку на левый виджет
$paned -> add1($image1);

# вторую на правыйs
$paned -> add2($image2);

# на базовое окно
$window -> add($paned);

# показываем
$window -> show_all();
Gtk3->main();
```

Методы виджета Paned

Чтобы получить два вертикально выровненных виджета panes надо использовать *"vertical"* вместо *"horizontal"*. Метод *add1($widget1)* добавит $*widget1* вверх, и *add2($widget2)* добавит $*widget2* вниз виджета, т.е. сделает набор дочерних виджетов либо горизонтальным, либо вертикальным.

5. Сигналы, вызовы и кнопки

5. 1. Сигналы, вызовы (to do)

5. 2. Кнопки

Виджет кнопки передает вызов в функцию.

Код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

# основное окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('GNOME Button');
$window->set_default_size(250,50);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# кнопка
my $button = Gtk3::Button->new();
# надпись, заголовок на кнопке, как чайлд, где парент - кнопка
$button->set_label('Click me');

# при нажатии послать сигнал 'clicked' функции do_clicked
$button->signal_connect('clicked' => \&do_clicked);

# добавить кнопку с тектом-заголовком на основную панель окно программы
$window -> add ($button);

# запустить все
$window -> show_all();
Gtk3->main();

# вызов функции по сигналу 'clicked' , пришедшему по нажатию кнопки
sub do_clicked {
        print("You clicked me! \n");
        }
```

методы виджета Button

В 20-й строке 'clicked' вызывает функцию do_clicked() при помощи строчки *$widget->signal_connect(signal, callback function)*. Смотри сигналы и вызовы для более детального объяснения.

- *set_relief("none")* отключает рельеф краев кнопки Gtk.Button – обычно находится в

состоянии *"normal"*.

- Если на кнопке будет набор иконок, выставить *set_use_stock(TRUE)* как признак набора иконок(перевести заново – не понял как сказать по русски). If the label of the button is a stock icon, *set_use_stock(TRUE)* sets the label as the name of the corresponding stock icon.
- Поместить на кнопку картинку (как иконку - ???) :

```
my $image = Gtk3::Image->new();
# USAGE: set_from_stock(stock-id, Gtk3::IconSize[3]);
$image -> set_from_stock("gtk-about", "4");
$button-> set_image($image);
```

- Если используется *set_focus_on_click(FALSE)* - кнопка не будет захватывать фокус ввода с мышки при нажатии оной на кнопку(подобно перемещению tab с клавиатуры по элементам интерфейса). Это в основном используется при работе с графическим интерфейсом с клавиатурой без мышки(иногда удобнее и привычнее).

5. 3. Гиперссылка(LinkButton)

Пример ссылки как она вгылядит.

Код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

# окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('GNOME LinkButton');
$window->set_default_size(250,50);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# кнопка
my $button = Gtk3::LinkButton->new('http://live.gnome.org');
# с заголовком
$button->set_label('Link to GNOME live');

# добавить
$window -> add ($button);
```

3 Unfortunately the nichnames for Gtk3::IconSize seems not to work at the moment because Perl/Gtk3 expects a integer. For this reason (I believe!) you have to use the following options:
'0' = 'invalid', '1' = 'menu', '2' = 'small-toolbar', '3' = 'large-toolbar, '4' = 'button', '5' = 'dnd' '6' = 'dialog'

```
# показать и запустить
$window -> show_all();
Gtk3->main();
```

методы для виджета (LinkButton widget)

- *get_visited()* возвращает *'visited'* состояние (*TRUE* или *FALSE*) если URI where the LinkButton points. Кнопка становится активной, когда по ней щелкают.
- *set_visited(TRUE)* становится *'visited'* состоянием URI когда LinkButton возвращает *TRUE* (аналогично для *FALSE*).
- При нажатии кнопки всякий раз идет сигнал *'activate-link'*. Для более деатального объяснения взаимодействия сигналов и перехватывающих их функций, смотрите раздел Signals и callbacks.

5. 4. Кнопка галка(CheckButton)

Кнопка нажата.

Код

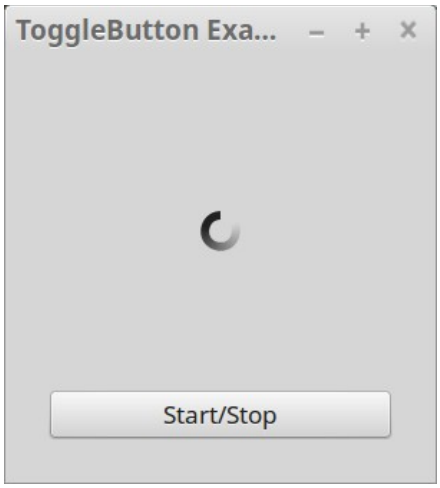
```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

# базовое окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('CheckButton Example');
$window->set_default_size(300,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# кнопка
my $button = Gtk3::CheckButton->new();
# с названием на ней
$button->set_label('Show Title');

# вызывает сигнал 'toggled' при нажатии который перехватывается функцциией toggled_cb
$button->signal_connect('toggled' => \&toggled_cb);

# по умолчанию чекбокс активен
$button->set_active(TRUE);
```

```
#  добавили
$window -> add($button);

# показали
$window -> show_all();
Gtk3->main();

sub toggled_cb {
        # если активно – показали заголовок окна
        # как 'Checkbutton Example'
        if ($button->get_active()) {
                $window->set_title ('CheckButton Example');
                }
        else {
         # иначе пустой
                $window->set_title ('');
                }
        }
```

Useful methods for a CheckButton widget

В строке 18 *'toggled'* вызывает функцию *toggled_cb()* используя следующий синтаксис *$widget->signal_connect("signal", \&callback_function)*. Смотри подробнее Signals и callbacks для более детального ознакомления.

5. 5. Кнопки изменения состояния(ToggleButton)



Когда нажата – крутиться анимированный индикатор загрузки.

код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# базовое окно
my $window = Gtk3::Window->new('toplevel');
```

```
$window->set_title ('ToggleButton Example');
$window->set_default_size(300,300);
$window->set_border_width(30);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# крутящийся анимированный индикатор
my $spinner = Gtk3::Spinner→new();

# с большим горизонтальным и вертикальным отступами
$spinner->set_hexpand(TRUE);
$spinner->set_vexpand(TRUE);

# кнопка
my $button = Gtk3::ToggleButton->new_with_label('Start/Stop');
# пускаем  сигнал 'toggled' если состояние виджета поменялось в функцию toggled_cb
$button->signal_connect('toggled' => \&toggled_cb);

# ставим виджеты на сетку
my $grid = Gtk3::Grid->new();
$grid->set_row_homogeneous(FALSE);
$grid->set_row_spacing(15);
$grid->attach($spinner, 0, 0, 1, 1);
$grid->attach($button, 0, 1, 1, 1);

# добавляем сетку на базовое окно
$window->add($grid);

# показали запустили
$window -> show_all();
Gtk3->main();

# вызов функции при получении сигнала 'toggled'
sub toggled_cb {
        # если togglebutton активна, запускаем вращение
        if ($button->get_active()) {
                $spinner->start();
                }
        # иначе тормозим его
        else    {
                $spinner->stop();
                }
        }
```

методы виджета ToggleButton widget

В 25 строке *'toggled'* сигнал вызывает функцию *toggled_cb()* используюя такую конструкцию *$widget->signal_connect("signal", \&callback_function)*. См. подробнее Signals и callbacks.

5. 6. Переключатели

Положение переключателя заставляет появиться или исчезнуть в заголовке окна текст.

код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('Switch Example');
$window->set_default_size(300,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# переключатель
my $switch = Gtk3::Switch->new();
# вернуть в начальное состояние
$switch->set_active(TRUE);
# соединиться со службой notify::active ,которая вызывает функцию activate_cb по
# срабатыванию переключателя
$switch->signal_connect('notify::active' => \&activate_cb);

# название
my $label = Gtk3::Label->new();
$label->set_text('Title');

# положить виджеты на сетку
my $grid = Gtk3::Grid->new();
$grid->set_column_spacing(10);
$grid->attach($label, 0, 0 , 1, 1);
$grid->attach($switch, 1, 0 , 1, 1);

# а сетку на базовое окно
$window->add($grid);

# показать все
$window -> show_all();
Gtk3->main();

# Вызов функции. До вызова notify::activv аргумент долженбыть активным
```

```
sub activate_cb {
	if ($switch->get_active) {
		$window->set_title('Switch Example');
		}
	else {
		$window->set_title('');
		}
	}
```

Методы виджета Switch

В 20-й строке сигнал службы *'notify::active'* вызывает функцию activate_*cb()* при помощи строки *$widget->signal_connect("signal", \&callback_function)*. См. подробнее Signals и callbacks.

5. 7. Кнопки выбора или радиокнопки(RadioButton)

Три кнопки, в терминале что-то выводят.

Код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('RadioButton Example');
$window->set_default_size(250,100);
$window->set_border_width(20);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# текст заголовок к кнопке 1
my $button1 = Gtk3::RadioButton->new('Button1');
$button1->set_label('Button 1');
# отправить сигнал 'toggled' по нажатию и вызвать функцию toggled_cb
$button1->signal_connect('toggled' => \&toggled_cb);

# текст заголовок к кнопке 2
my $button2 = Gtk3::RadioButton->new_from_widget($button1);
$button2->set_label('Button 2');
# отправить сигнал 'toggled' по нажатию и вызвать функцию toggled_cb
$button2->signal_connect('toggled' => \&toggled_cb);
```

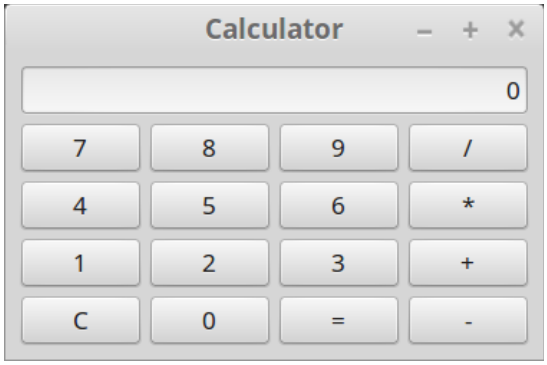
```
# по умолчанию не выделять радиокнопку 2
$button2->set_active(FALSE);

# для третьей кнопки поступить аналогично второй
my $button3 = Gtk3::RadioButton->new_from_widget($button1);
$button3->set_label('Button 3');
$button3->signal_connect('toggled' => \&toggled_cb);
$button3->set_active(FALSE);

# добавить на сетку кнопки
my $grid = Gtk3::Grid->new();
$grid->attach($button1, 0, 0, 1, 1);
$grid->attach($button2, 0, 1, 1, 1);
$grid->attach($button3, 0, 2, 1, 1);
# добавть сетку на базовое окно
$window->add($grid);

# показать все
$window -> show_all();
Gtk3->main();

# вызов обработчика события выделения кнопки
sub toggled_cb {
	# первое значение переменной , переданной в функцию (@_) всегда содержит ссылку
	# на виджет, который послал этот сигнал
	my ($button) = @_;
	# строка, чтобы описать состояние кнопки
	my $state = 'unknown';
	# каждый раз, когда кнопка включена, состояние on
	if ($button->get_active()) {
		$state = 'on';
		}
	# иначе off
	else {
		$state = 'off';
		}
	# при всяком вызове функции (кнопка или on или off)
	# печатается в терминале что происходит: on/off
	my $button_label = $button->get_label();
	print ("$button_label was turned $state \n");
	}
```

Useful methods for a RadioButton widget

В 20-й строке сигнал *'toggled'* вызывает функцию toggle*_cb()* при помощи строки *$widget->signal_connect("signal", \&callback_function)*. См. подробнее Signals и callbacks.

Другим образом создать радио кнопку с названием можно при помощи строчки *"my $button1 = Gtk3::RadioButton->new_with_label("", "Button1");"*. (первый аргумент это группа радиокнопок, которая может быть определена с помощью функции *get_group()*, второй аргумент – название кнопки.

6. ButtonBox



Калькулятор – кнопки включены в горизонтальную группу ButtonBoxes.

Код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('Calculator');
$window->set_default_size(350,200);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# поле ввода
my $entry = Gtk3::Entry->new();
# с предопределенным текстом '0'
$entry->set_text('0');
# текст выравнен в правую сторону
$entry->set_alignment(1);
# the text in the entry cannot be modifier writing in it
$entry->set_can_focus(FALSE);

# сетка
my $grid = Gtk3::Grid->new();
$grid->set_row_spacing(5);

# ставим на сетку поле ввода
$grid->attach($entry, 0,0,1,1);

# задаем надписи на кнопках
my @buttons =          (7, 8, 9, '/',
                        4, 5, 6, '*',
                        1, 2, 3, '+',
                        'C', 0, '=', '-');

# для каждой строки ButtonBox, прикрепляем на сетку
foreach my $i (0..3) {
```

```
	my $hbox = Gtk3::ButtonBox->new('horizontal');
	$hbox->set_spacing(5);
	$grid->attach($hbox, 0, $i+1, 1, 1);
	# каждый ButtonBox имеет 4 кнопки, вызывающиее соответствующую функцию
	foreach my $j (0..3) {
		my $button=Gtk3::Button->new();
		$button->set_label("$buttons[$i*4+$j]");
		$button->set_can_focus(FALSE);
		$button->signal_connect('clicked' => \&button_clicked);
		$hbox->add($button);
		}
	}

# переменные для расчета
my $first_number = 0;
my $second_number = 0;
my $counter = 0;
my $operation = '';

# виджет сетки с виджетами на ней добавляется на основное окно программы
$window->add($grid);

# запустить все
$window -> show_all();
Gtk3->main();

# функция или подпрограмма для обработки нажатия кнопки
sub button_clicked {
	# читаем содержимое сигнала с кнопки
	my $button = $_[0];

	#  берем текст с кнопки
	my $label = $button->get_label();

	# операции
	if ($label eq '+') {
		$counter += 1;
		if ($counter > 1) {
			do_operation();
			}
		$entry->set_text('0');
		$operation = 'plus';
		}
	elsif ($label eq '-') {
		$counter += 1;
		if ($counter > 1) {
			do_operation();
			}
		$entry->set_text('0');
		$operation = 'minus';
		}
	elsif ($label eq '*') {
```

```
                $counter += 1;
                if ($counter > 1) {
                        do_operation();
                        }
                $entry->set_text('0');
                $operation = 'multiplication';
                }
        elsif ($label eq '/') {
                $counter += 1;
                if ($counter > 1) {
                        do_operation();
                        }
                $entry->set_text('0');
                $operation = 'division';
                }
         # for '='
        elsif ($label eq '=') {
                do_operation();
                $entry->set_text("$first_number");
                $counter = 1;
                }
        # для выхода
        elsif ($label eq 'C') {
                $first_number = 0;
                $second_number = 0;
                $counter = 0;
                $entry->set_text('0');
                $operation = '';
                }
        # показ чисел
        else {
                my $new_digit = $button->get_label();
                # ZERO DIVISI ERROR -> TO DO!!!
                my $number = $entry->get_text();
                $number = $number * 10 + $new_digit;
                if ($counter eq '0') {
                        $first_number = $number;
                        }
                else {
                        $second_number = $number;
                        }

                $entry->set_text("$number");
                }
        }

sub do_operation {
        if ($operation eq 'plus') {
        $first_number += $second_number;
                }
        elsif ($operation eq 'minus') {
        $first_number -= $second_number;
```

```
        }
    elsif ($operation eq 'multiplication') {
    $first_number *= $second_number;
        }
    elsif ($operation eq 'division') {
    $first_number = $first_number / $second_number;
    # ZERO DIVISI ERROR -> TO DO!!!
        }
    }
```

Используемые методы для виджета ButtonBox

- Компоновка ButtonBox определяется функцией *set_layout(layout)*, где аргумент может иметь значения *"spread"* (кнопки равномерно расположены по полю), *"edge"* (кнопки располагаются по краям), *"start"* (кнопки сгруппированы в начале области ButtonBox), *"end"* (кнопки сгруппированы в конце области ButtonBox), "*center*" (кнопки центрируются), "expand" (кнопки заполняют все поле).
- *Фенкция set_child_secondary($button, TRUE)* определяет должна ли кнопка появиться во вторичной группе дочерних элементов. Типичное использование вторичного дочернего элемента - кнопка справки в диалоговом окне. Эта группа появляется после других дочерних элементов если компоновка ButtonBox вида *"start"*, "spread" or *"edge"*, и перед дочерними элементами если параметр *set_layout ()* - *"end"*. Если вид ButtonBox установлен в положение *"start"* или *"end"*, дочерние элементы выстраиваются в другом конце поля. Для других стилей отображения они появляются сразу после родительских элементов.
- *Функция set_child_non_homogeneous($button, TRUE)* устанавливает освобожден ли дочерний виджет от подстройки под однотипные размеры. По умолчанию FALSE.
- *set_spacing(spacing)* устанавливает число пикселов между кнопками или элементами виджета.

7. Другие отображаемые элементы

7. 1. Statusbar

statusbar говорит, если нажата кнопка или какая-либо клавиша на клавиатуре

Код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
```

```
use Glib ('TRUE','FALSE');

# окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('StatusBar Example');
$window->set_default_size(200,100);
# ввод с клавиатуры
$window->signal_connect('key-press-event' => \&do_key_press_event);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# название
my $label = Gtk3::Label->new('Press any key or ');

# кнопка
my $button = Gtk3::Button->new('click me.');
# вызвать обработчик сигнала нажатия кнопки
$button->signal_connect('clicked', \&button_clicked_cb);

# состояние кнопки
my $statusbar = Gtk3::Statusbar->new();
# переменная $context_id – не показывается в пользовательском интерфейсе, но нужна
# для определния источника сигнала
my $context_id = $statusbar->get_context_id('example');
# передать сообщение в очередь статусбара
$statusbar->push($context_id, 'Waiting for you to do something...');

# наложить все на сетку, расположить виджеты по координатам на базовом окне
my $grid = Gtk3::Grid->new();
$grid->set_column_spacing(5);
$grid->set_column_homogeneous(TRUE);
$grid->set_row_homogeneous(TRUE);
$grid->attach($label, 0, 0, 1, 1);
$grid->attach_next_to($button, $label, 'right', 1, 1);
$grid->attach($statusbar, 0, 1, 2, 1);

# сетку на базовое окно
$window->add($grid);

# показать все
$window -> show_all();
Gtk3->main();

# вызов функции при нажатии кнопки
# если кнопка нажата, событие нажатия кнопки передается статусбару
# который меняет свое значение
sub button_clicked_cb {
        $statusbar->push($context_id, 'You clicked the button.');
        }

# обработчик событий с клавиатуры сообщает статусбару символьное имя кнопки

sub do_key_press_event {
```

```
    my ($widget, $event) = @_;
    my $keyval = $event->keyval;
    # !!! запомни: perl-вариант Gtk3::Gdk->keyval_name($event);
    # не работает  !!!
    my $key = Gtk3::Gdk::keyval_name($keyval);
    $statusbar->push($context_id, "$key key was pressed.");

    # остановить сигнал
    return TRUE;
    }
```

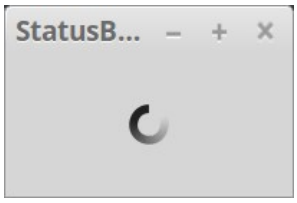
Gtk3::Gdk::keyval_name($keyval)[4] конвертирует $event->keyval в символическое имя. Имена и отвечаюшие им наименования клавишь находятся here, исключая кнопку GDK_KEY_BackSpace возвращающую строку 'BackSpace'.

Основные методы виджета Statusbar

В 21-й строке сигнал *'clicked'* вызывает функцию *button_clicked_cb()* при помощи строки *$widget->signal_connect("signal", \&callback_function)*. См. подробнее Signals и callbacks.

- *pop($context_id)* удаляет первое сообщение в очереди(стеке) сообщений с заданным context_id
- *remove_all($context_id)* удаляем все сообщения из стека сообщений с заданным *context_id*
- *remove($context_id, $message_id)* удаляет сообщение *message_id* из очереди сигналов с заданным *context_id*. *message_id* возвращает message_id = *push($context_id, "the message")* при продвижении сообщения в стеке .

7. 2 Spinner



Spinner начинает и заканчивает крутиться при нажатии пробела.

код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# окно
my $window = Gtk3::Window->new('toplevel');
```

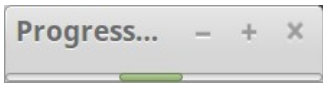
4 The more perlish Schreibweise *"Gtk3::Gdk->keyval_name"* seems not to work correctely

```
$window->set_title ('StatusBar Example');
$window->set_default_size(200,100);
$window->signal_connect('key-press-event' => \&do_key_press_event);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# крутилка
my $spinner = Gtk3::Spinner->new();
# включаем
$spinner -> start();
# добавляем на окно
$window->add($spinner);

# показать
$window -> show_all();
Gtk3->main();

# обработчик сигнала с клавиатуры(пробела), который запускает/останавливает spinner

sub do_key_press_event {
	# $keyname символическое имя сигнала по нажатию кнопки клавиатуры
	my ($widget, $event) = @_;
	my $keyval = $event->keyval;
	# !!! IMPORTANT: The perlish diction Gtk3::Gdk->->keyval_name($event);
	# doesn't work !!!
	my $keyname = Gtk3::Gdk::keyval_name($keyval);

	# если пробел( 'space' )
	if ($keyname == 'space') {
		# запустить  крутилку
		if ($spinner->get_property('active')) {
			# остановить
			$spinner->stop();
			}
		# есль неактивно
		else {
			# запустить снова
			$spinner->start();
			}
		}
	# остановить сигнал
	return TRUE;
}
```

Gtk3::Gdk::keyval_name($keyval)[5] конвертирует $event->keyval в символическое имя. Имена и отвечающие им наименования клавиш находятся here, исключая кнопку GDK_KEY_BackSpace возвращающую строку 'BackSpace'.

5 The more perlish Schreibweise *"Gtk3::Gdk->keyval_name"* seems not to work correctely

7. 3. ProgressBar



ProgressBar запускается и останавливается по нажатию той или иной клавиши.

код

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('ProgressBar Example');
$window->set_default_size(220,20);
$window->signal_connect('key-press-event' => \&do_key_press_event);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# прогрессбар
my $progressbar = Gtk3::ProgressBar->new();
# добавить его на окно
$window->add($progressbar);

# вызывается каждые  100 миллисекунд и $source_id берет ID события
# (т.е. полоска прокрути меняется каждые 100 миллисекунд)
my $source_id = Glib::Timeout->add(100, \&pulse);

# показать
$window -> show_all();
Gtk3->main();

# обработчик сигнала с клавиатуры, который запускает/останавливает progressbar
sub do_key_press_event {
	# если progressbar остановлен ( $source_id == 0 )
	#  то условием ниже он включается
	if ($source_id == 0) {
		$source_id = Glib::Timeout->add(100, \&pulse);
		}
	# если выключен, то удаляется из очереди сообщений
	else {
		Glib::Source->remove($source_id);
		$source_id = 0;
		}
	# остановить все
	return TRUE;
	}
```

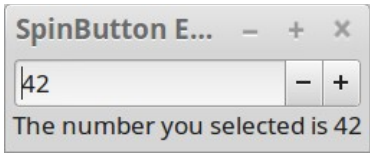
```
# сделать прогрессбар актвным
sub pulse {
	$progressbar->pulse();
	# вызвать функцию
	return TRUE;
	}
```

Основные методы виджета ProgressBar

- Вместо функции *pulse()*, которая заставляет двигаться прогрессбар туда обратно, можно заставить его постепенно заполняться с помощью функции *set_fraction(fraction)*
- Для показывания текста, наложеного на панель , ипользуется функция *set_text("text")* и *set_show_text(TRUE)*. Если текст не задан и s*et_show_text(TRUE)* текст будет показываться в зависимости от величины заполнения прогрессбара.

8. Вдижеты ввода

8. 1. Инкремент/декремент(SpinButton)



Поволяте выбрать число, либо введя его, либо нажатием кнопок +/- на клавиатуре

Пример использования

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# базовая панель
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('SpinButton Example');
$window->set_default_size(210,70);
$window->set_border_width(5);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});
```

```
# задаем числовые диапазоны “an adjustment (initial value, max value,
# step increment - press cursor keys or +/- buttons to see!,
# page increment  not used here,
# page size - not used here)”
my $ad = Gtk3::Adjustment->new(0, 0, 100, 1, 0, 0);

# кнопк для числовых (digits=0)
# значений ($adjustment, climb_rate, digits)
my $spin = Gtk3::SpinButton->new($ad, 1, 0);
# на всю длину виджета
$spin->set_hexpand(TRUE);

# получение сигнала 'value-changed' ведет к вызову функции spin_selected
$spin->signal_connect('value-changed' => \&spin_selected);

# заголовок
my $label = Gtk3::Label->new();
$label -> set_text('Choose a number');

# сетка
my $grid = Gtk3::Grid->new();
$grid->attach($spin, 0, 0, 1, 1);
$grid->attach($label, 0, 1, 2, 1);

# добавляем сетку на базовую панель
$window->add($grid);

# показать все
$window -> show_all();
Gtk3->main();

# изменение значения вызывает изменение текста
sub spin_selected {
        my $number = $spin->get_value_as_int();
        $label->set_text("The number you selected is $number");
        }
```

Основные методы виджета SpinButton

Метод Gtk3::Adjustment необходим для создания Gtk3::SpinButton. Он определяет размер и относительное положение(нижняя и верхняя граница, шаг и инкремент/декремент) виджета на основной панели. Определяется как *Gtk3::Adjustment->new(value, lower, upper, step_increment, page_increment, page_size)*, где значения имеют тип флоат, *step_increment* это инкремент/декремент, который зменяется в результате действий пользователя мышкой или кнопками на клавиатуре. Обратите внимание, что в этом примере *page_increment* и *page_size* не используются в этом примере и должны быть установлены в 0.

В строке 27 сигнал *'value-changed'* вызывает функцию *spin_selected()* с помощью $widget->signal_connect("signal" => \&callback function). Смотрите секцию Signals и Callbacks для

более детального обзора.

- Если вы хотите, чтобы значение spinbuttonповторилось снова при превышении максимльного значения, установите функцию set *set_wrap(TRUE)*. Сигнал 'wrapped' приведет состояние счетчика в начальное значение.
- *set_digits(digits)* устанавливает количество цифр, которое может быть показано spinbutton (до 20 знаков).
- Чтобы получить значение spinbutton как целого числа, используйте *get_value_as_int()*.

8. 2. Виджет ввода(Entry)

Данное приложение приветствует пользователя в терминале введенным им именем

Code used to generate this example

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# основная панель
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('What is your name?');
$window->set_default_size(300,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# одна строка ввода
my $name_box = Gtk3::Entry->new();
# вызывает сигнал, если нажата кнопка Enter, который обрабатывается
# функцией cb_activate
$name_box->signal_connect('activate', \&cb_activate);

# добавить на основную панель
$window->add($name_box);

# показать все и запустить
$window -> show_all();
Gtk3->main();

# содержимое вижета ввода, показывается в терминале
sub cb_activate {
        # доставка соержимого виджета
```

```
my $entry = $_[0];
my $name = $entry->get_text();
# напечатать в терминале

print "Hello $name! \n";
}
```

Основные методы виджета Entry

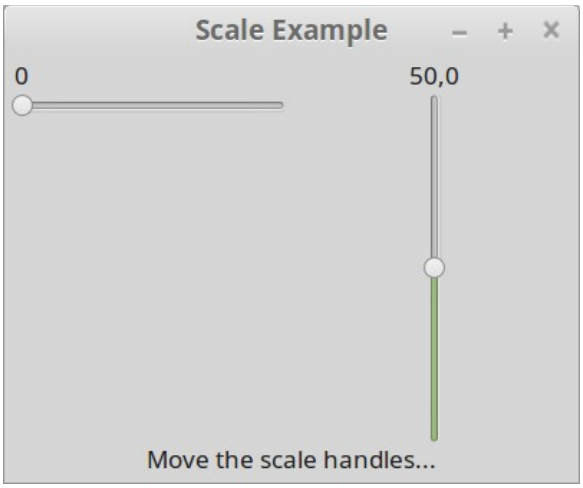
В 12 строке сигнал *'activate'* вызывает функцию *cb_activate()* используя *$widget->signal_connect("signal", \&callback function)*. См Сигналы и вызовы для более подробного ознакомления. . Некоторые сигналы которые испольует виджет *Gtk3::Entry* : *'activate'* (вызывается, когда пользователь нажал кнопку ввода); *'backspace'* (вызывается, когда пользователь нажал кнопки Backspace или Shift-Backspace); *'copy-clipboard'* (Ctrl-c и Ctrl-Insert); *'paste-clipboard'* (Ctrl-v и Shift-Insert); *'delete-from-cursor'* (удаление текста, с позиции курсора; Ctrl-Delete, для удаления слова); *'icon-press'* (вызывается, когда пользователь нажимает по соответствующей иконке); *'icon-release'* (вызывается, когда пользователь отпускает мышку на соответствующей иконке); *'insert-at-cursor'* (вставка текста в положение, заданное курсором в поле ввода); *'move-cursor'* (отслеживает движение курсора); *'populate-popup'* (вызывается перед показом контекстного меню или подсказки, если необходимо).

- *get_buffer()* и *set_buffer($buffer)*, где $buffer - объект Gtk3::EntryBuffer , который можно установить, чтобы брать или вставлять данные из буфера.
- *get_text()* и *set_text('some text')* взять и установить содержимое виджета ввода.
- *get_text_length()* длинна текста.
- *get_text_area()* область показа текста.
- Если установлена *set_visibility(FALSE)* Если установлена *set_visibility(FALSE)* символы в виджте не отображаются , но существуют. Это лучший путь для отображения невидимых символов текущего фонта. Сие извращение можно изменить с помощью функции *set_invisible_char(ch)* где ch символ в Юникоде. Инверсия последнего метода достигается функцией *unset_invisbile_char()*- (оставляю перевод на волю читателя ибо не понял назначения таких невидимых текстов - the characters in the entry are displayed as the invisible char. This is the best available invisible character in the current font, but it can be changed with *set_invisible_char(ch)*, where ch is a Unicode charcater. The latter method is reversed by *unset_invisbile_char())*.
- *set_max_length(int)*, ограничивает длину запеси, где аргумент int – число символов в поле ввода.
- По умолчанию, если вы нажали кнопку ввода, объект *Gtk3::Entry* испускает сигнал *'activate'*. Если хочеться сэмулировать нажатие кнопки воода по умолчанию в самом окне (используя *set_default($widget)*), установите *set_activates_default(TRUE)*.
- Установить фрейм фокруг виджета ввоа: *set_has_frame(TRUE)*.

- *set_placeholder_text('some text')* установить текст в виджете ввода, когда он находится не в фокусе(не активен, в смысле передвижения по элементам управления с клавиатуры кнопкой табуляция а после стрелочками) .
- *set_overwrite_mode(TRUE)* и *set_overwrite_mode(FALSE)* разрешать изи запрещать перезапись.
- Если *set_editable(FALSE)* пользователь не может редактировать текст в виджете.
- *set_completion($completion)*, где *$completion - объект Gtk3::EntryCompletion*, покаывает заполнение – или нет, если *$completion* - 'None'.
- Виджет ввода может отслеживать степень заполнения поля ввода при помощи функции *set_progress_fraction(fraction)*, где заполнение меняется от 0.0 до 1.0 включительно, постепенно аполняя поле. Мы используем функцию *set_progress_pulse_step()* для отслеживания степени заполнения виджета ввода от всей части посредством вызова *progress_pulse()*. Можно также установить величину шага инкремента декремента при помощи *set_progress_pulse_step()*.
- Видет ввода также може показывать иконки, которые могут обладать свойством, например, подсказок.

  Для добавления иконки испольуются функция *set_icon_from_stock(icon_position, stock_id)*, или одна из нижеследующих функций *set_icon_from_pixbuf(icon_position, pixbuf)*, *set_icon_from_icon_name(icon_position, icon_name)*, где *icon_position* принимает значения *"primary"*[6] (установить в начало ввода) *"secondary"*[7] (установить в конец ввода). Для установки всплывающей подсказки используйте функцию *set_icon_tooltip_text('tooltip text')* или *set_icon_tooltip_markup('tooltip text in Pango markup language')*.

8. 3. Виджет прокрутки(Scale)



Прокрутка слайдов!!

6 Or Gtk3::EntryIconPosition::PRIMARY
7 Or Gtk3::EntryIconPosition::SECONDARY

Код для этого примера

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# базовая панель
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('Scale Example');
$window->set_default_size(400,300);
$window->set_border_width(5);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# два вида позиционирования (initial value, min value, max value,
# шаг прокрутки – жвакайте клавишами и смотрите что будет!,
#  - двигайте стрелки the handle to see!,
# размер страницы не используется в данном примере)
my $ad1 = Gtk3::Adjustment->new(0,0,100,5,10,0);
my $ad2 = Gtk3::Adjustment->new(50,0,100,5,10,0);

# горизонтальный масштаб
my $h_scale = Gtk3::Scale->new('horizontal',$ad1);
# или в цифровом виде (без символов)
$h_scale->set_digits(0);
# который может расшириться горизонтально, если есть пространство
# на сетке (см. ниже)
$h_scale->set_hexpand(TRUE);
# выравнивание сверху
# (см. ниже)
$h_scale->set_valign('start');

# обработка сигнала 'value-changed' масштабирования
# функцией scale_moved
$h_scale->signal_connect('value-changed' => \&scale_moved);

# вертикальное масштабирование
my $v_scale = Gtk3::Scale->new('vertical',$ad2);
# вертиакальное расширение по сетке
# (см. ниже)
$v_scale->set_vexpand(TRUE);

# вызов соответствующей функции по сгналу 'value-changed'

$v_scale->signal_connect('value-changed' => \&scale_moved);

# название
my $label = Gtk3::Label->new();
$label->set_text('Move the scale handles...');

# ставим на сетку
my $grid = Gtk3::Grid->new();
```

```
$grid ->set_column_spacing(10);
$grid->set_column_homogeneous(TRUE);
$grid->attach($h_scale, 0, 0, 1, 1);
$grid->attach_next_to($v_scale, $h_scale, 'right', 1, 1);
$grid->attach($label, 0, 1, 2, 1);

$window->add($grid);

# поместить все на базовое окно и запустить приложение
$window -> show_all();
Gtk3->main();

# обработка сигналов масштабирования для отображение в тексте
#  если было изменение
sub scale_moved {
        my ($widget, $event) = @_;
        my $h_value = $h_scale->get_value();
        my $v_value = $v_scale->get_value();
        $label->set_text("Horizontal scale is $h_value; vertical scale is $v_value.");
        }
```

Используемые методы для виджета Scale

Позиционирование *Gtk3::Adjustment* необходимо для построения *Gtk3::Scale*. Подстраивание рамера определяется функцией *Gtk3::Adjustment->new(value, lower, upper, step_increment, page_increment, page_size)* в которой значения могут иметь тип float; *step_increment* - инкремент/декремент, получаемый от клавиш управления, *page_increment* – размер масштаба. Отметим, что *page_size* не используется в данном примере и должна быть установлена в 0.

В строке 34 сигнал *'value-changed'* обрабатывается ункцией *scale_moved()* при помощи *$widget->signal_connect("signal" => \&callback function)*. см. Сгналы и вызовы для подробного изучения.

- *get_value()* возвращает текущее начение масштаба; *set_value(value)* устанавливает его (если то значение с плаваюе точкой будет вне максимального или минимального диапазона, оно будет автоматически установлено в соответствующее отображению начение). Таковы методы класса Gtk3::Range.
- Чтобы не отображать текуее значение около полунка, используйте функцию *set_draw_value(FALSE)*.
- Для соответствия масштаба оригинала отображению испольуйте функции (не тестировалось):

```
$h_scale->set_restrict_to_fill_level(FALSE);
$h_scale->set_fill_level($h_scale->get_value());
$h_scale->set_show_fill_level(TRUE);
```

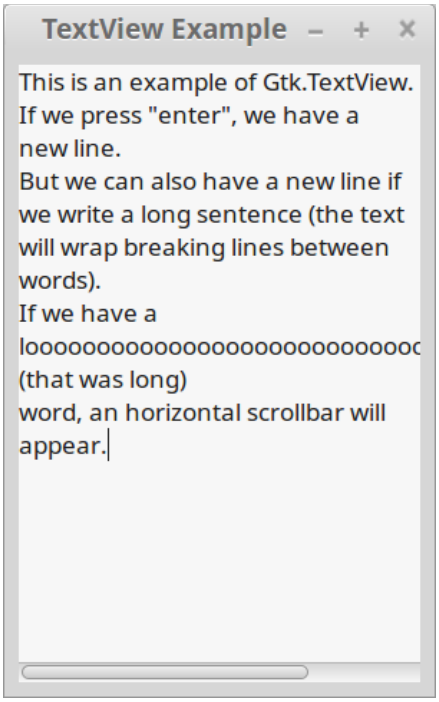
при вызове функции *'value-changed'* signal, чтобы оперативно отслеживать начение

переменной коэффциента масштабируемости. Так работают методы класса Gtk3::Range.

- *add_mark(value, position, markup)* добавляет метку *value* (число с плавающей точкой или целое, ависит от точности масштабирования), на *position* ("left", "right", "top", "bottom") с текстом *Null* или *markup* в Pango Markup Language. Для удаления метки вызвается, *clear_marks()*.
- *set_digits(digits)* установить точность масштабирования.

9. Виджет для написания текста

9. 1. Виджет TextView



Код для того примера

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# баовая панель
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('TextView Example');
$window->set_default_size(300, 450);
$window->set_border_width(5);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# скроллбар прокртук для дочернего виджета (that is going to be the textview!)
my $scrolled_window = Gtk3::ScrolledWindow->new();
```

```
$scrolled_window->set_border_width(5);
# ксроллим, прокручиваем, если надо
$scrolled_window->set_policy('automatic','automatic');

# буферизуем, запоминаем текст
my $buffer1 = Gtk3::TextBuffer->new();

# сам виджет текста
my $textview = Gtk3::TextView->new();
# отображаем буфер
$textview->set_buffer($buffer1);
# перенести слова на новые строки, если надо
$textview->set_wrap_mode('word');

# прокручиваем
$scrolled_window->add($textview);

$window->add($scrolled_window);

# показываем
$window -> show_all();
Gtk3->main();
```

Оспользуемые методы виджета TextView

Gtk3::TextView показывает текст находящийся в *Gtk3::TextBuffer*. Однако, большая часть манипуляцй с текстом происходит с помощью *Gtk3::TextIter*[8] - позиция между двумя символами в буфере. Повторения расстояний между символами не могут быть неопределенными, всякий раз при изменении буфера с текстом, т.е. самого текста, поля между буквами переопределяются во всем буфере И а этого повторения нельзя использовать для сохранения позиций текста в буфере. Чтобы зафиксировать позиционирование текста, испольуется *Gtk3::TextMark*, который можно установить в значение *visible(TRUE)*. Текстовый буфер содержит две служебные метки - '*insert*' (поиция курсора) и *'selection_bound'* (нечо вроде общего подстраивания текста под размер виджета ввода - ??).

Методы виджета TextView :

- Виджет TextViewпо умолчанию редактируемый. Если хотите иначе – используйте *set_editable(FALSE)*. Если в буфере нет редактируемого текста. Это может быть хорошей идеей для использования функции *set_cursor_visible(FALSE)*.
- Выравнивание текста в форме определяется функцией *set_justification(Gtk.Justification)* где *Gtk3.Justification* имеет значения *"left"*, *"right"*, *"center"*, *"fill"*.
- Перено строки в виджете задается функцией s*et_wrap_mode(Gtk.WrapMode)* где *Gtk.WrapMode* может быть *"none"* (поле ввода большое), *"char"* (перенос строк

8 A Gtk3::TextIter is initialized with the $textbuffer->get_iter_* methods (for more informations see the API Reference)

побуквенно), *"word"* (перенос строк по словам), *"word_char"* (перенос строк между словами, если это не мешает переносу побуквенно –??).

Методы виджета TextBuffer :

- *get_insert()* возвращает *Gtk3::TextMark* позицию курсора в точке ввода.
- *get_selection_bound()* позвращает *Gtk3::TextMark* связанного выбора –???.
- *set_text('some text', length)* где *length* положительное число или -1, устанавливает содержмое буфера как длинну некоторого заданного текста. Если *length* не задана или равна -1, текст занимает все поле и содержимое буфера уничтожается.
- *insert($iter, 'some text', length)* где *$iter* повторение расстояния между буквами и переменная *length* либо положительна, либо -1, эти величины определяют как будет втавляться текст в форму, подстраивая межбуквенный интервал, позицию курсора и т.п. под рамер формы в зависимости от количесва текста.
- *insert_at_cursor('some text', length)* делает тоже самое, что и *insert($iter, 'some text', length)*, с текущим положением курсора и его итерацией.
- *create_mark('mark_name', $iter, left_gravity)* где второй аргумент позиционирования *Gtk3::TextIter* и *left_gravity* булева переменная, задающая *Gtk3::TextMark* позицию выравнивания текста. Если *'mark_name'* - None, отметка позиции текста анонимная; в противном случае возвращается имя метки *get_mark()*. Если метка *left gravity*, и текст втавляется относительно текущего положения курсора, метка перемещается влево о вставленного текста. Если *left_gravity* опущена, функция вовращает False.
- Чтобы определить, какое у некоторого текста в буфере должно быть форматирование, Вы должны определить тег, содержащий информацию о форматировании, и затем применить тот тег к области текста, используя *create_tag('tag name', property)* и *apply_tag(tag, start_iter, end_iter), как в, например, следующих строках*:

```
tag = $textbuffer->create_tag('orange_bg', background='orange');
$textbuffer->apply_tag(tag, $start_iter, $end_iter);
```

Ниже приведены некоторые общие стили, применяемые к тексту:

- Background colour ('background' property)
- Foreground colour ('foreground' property)
- Underline ('underline' property)
- Bold ('weight' property)
- Italics ('style' property)
- Strikethrough ('strikethrough' property)
- Justification ('justification' property)
- Size ('size' and 'size-points' properties)

- Text wrapping ('wrap-mode' property)

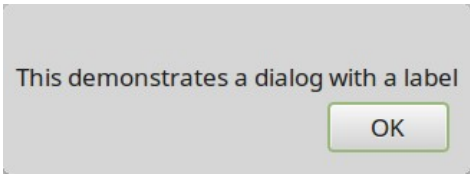
Вы можете также удалить определенные теги, используя *remove_tag()* или вообще убрав их при помощи вызова функции *remove_all_tags()*.

Методы виджета TextIter

- forward_search(needle, flags, limit) поиск на опережение для общего выравнивания текста. Поиск стиля показываемого текста не будет продолжаться вне значения установленного Gtk.TextIter. Флаги могут быть установлены как по отдельности так и в виде комбинации побитового или |: 0 (соответствие должно быть точным); Gtk.TextSearchFlags.VISIBLE_ONLY (невидимый текст также может обрабатываться для дальнейшего расчета форматирования); Gtk.TextSearchFlags.TEXT_ONLY (на положение текста могут оказывать другие дочерние виджеты); Gtk.TextSearchFlags.CASE_INSENSITIVE (распределение положения текста не зависит о регистра). Метод возвращает набор указателей Gtk.TextIter для начала поиционирования следующего текста; если ничего не найденовозвращается None.
- backward_search(needle, flags, limit) работает аналогично forward_search(), но в другую сторону.

10. Виджет диалога(Dialog)

10. 1. Диалог



Диалоговое окно, выывающее некоторую функцию по нажатию кнопки

Code used to generate this example

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

# базовое окно
my $window = Gtk3::Window->new('toplevel');
$window->set_title ('GNOME Button');
$window->set_default_size(250,50);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit()});

# кнопка
my $button = Gtk3::Button->new('Click me');
# обработка сигнала 'clicked' по нажатию кнопки функцией
# on_button_click()
$button->signal_connect('clicked', \&on_button_click);
```

```
# добавим кнопку на базовое окно
$window->add($button);

# запустить все
$window -> show_all();
Gtk3->main();

# вызов функции по приходу сигнала 'clicked' после нажатия кнопки на базовом окне

sub on_button_click {
	# создаем диалог
	my $dialog = Gtk3::Dialog->new();
	$dialog->set_title('A Gtk+ Dialog');

	# Окно, определенное в контрукторе ($window) является
	# родителем диалогового окна
	# Кроме того, диалоговое окно находится поверх родительского окна
	$dialog->set_transient_for($window);

	# установить правило: никаких взаимодействий с другими окнами
	$dialog->set_modal(TRUE);

	# добавить кнопку на диалоговое окно
	$dialog->add_button('OK','ok');

	# Передать 'response' сигнал (кнопка была нажата) в
	# функцию on_response()
	$dialog->signal_connect('response' => \&on_response);

	# Добавить на область действия диалога некоторый текст
	my $content_area = $dialog->get_content_area();
	my $label = Gtk3::Label->new('This demonstrates a dialog with a label');
	$content_area->add($label);

	# показать диалог
	$dialog->show_all();
	}

sub on_response {
	my ($widget, $response_id) = @_;
	print "response_id is $response_id \n";
	# убрать виджет диалога когда выывается функция on_response()
	# (т.е. когда кнопка в диалоговом окне была нажата)
	$widget->destroy();
	}
```

Useful methods for a Dialog widget

В строке 17 сигнал *'clicked'* вызывает функцию *on_button_click()* используя *$widget->signal_connect("signal" => \&callback function)*. См. сигналы и функции для более детального понимания.

- Вместо значения *set_modal(TRUE)* можно выставить значение *set_modal(FALSE)* которое придет в соответствие со значением *set_destroy_with_parent(TRUE)* в случае закрытия основного окна(криво перевел, перевести заново).
- *add_button(button_text, response_id)*, где *response_id* любое число, является альтернативой add_button(button_text, Gtk.ResponseType), где *Gtk.ResponseType* может иметь следующие значения "ok", "cancel", "close", "yes", *"no"*, *"apply"*, *"help"*, которые поочередно соответствуют целым числам -5, -6,..., -11.

10. 2. AboutDialog

Пример работы ваджета AboutDialog с использованием Gtk3::Window и Menu (диалог 'about' показывается если выбрать в меню 'About').

Code used to generate this example

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

my $window = Gtk3::Window->new('toplevel');
$window->set_title('AboutDialog Example');
$window->set_default_size(200,200);
$window->signal_connect('destroy'=>sub {Gtk3->main_quit;});

# выкидное меню создается в методе create_menubar (см. ниже)
my $menubar = create_menubar();

# добавить выкидное меню на сетку
my $grid=Gtk3::Grid->new();
$grid->attach($menubar,0,0,1,1);

# добавить сетку на базовое окно или базовую панель
$window->add($grid);

$window->show_all();
Gtk3->main();
```

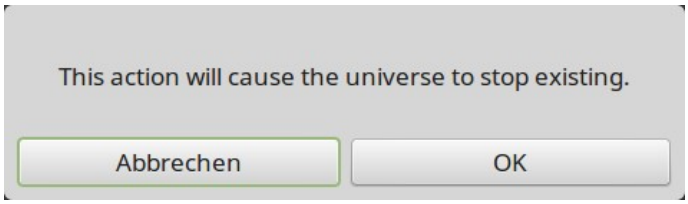
```
sub create_menubar {
	# создаем выкидное меню
	my $menubar=Gtk3::MenuBar->new();

	# создаем элемнт выкидного меню
	my $menubar_item=Gtk3::MenuItem->new('Anwendung');

	# добавляем элемент меню в выкидное меню
	$menubar->insert($menubar_item,0);

	# создаем меню(или список функций)
	my $menu=Gtk3::Menu->new();

	# добавляем 2 элемента в меню
	my $item1=Gtk3::MenuItem->new('About');
	$item1->signal_connect('activate'=>\&about_cb);

	my $item2=Gtk3::MenuItem->new('Quit');
	$item2->signal_connect('activate'=> sub {Gtk3->main_quit();});


	$menu->insert($item1,0);
	$menu->insert($item2,1);

	# добавляем все это в выкидное меню
	$menubar_item->set_submenu($menu);

	# возвращаем созданное меню
	return $menubar;
	}

sub about_cb {
	# используем Gtk3::AboutDialog
	my $aboutdialog = Gtk3::AboutDialog->new();

	# добавляем лист авторов  (будет использовано позднее)
	my @authors = ('GNOME Documentation Team');
	my @documenters = ('GNOME Documentation Team');

	# заполняем aboutdialog
	$aboutdialog->set_program_name('AboutDialog Example');
	$aboutdialog->set_copyright(
		"Copyright \xa9 2012 GNOME Documentation Team");
	# апомните,  set_authors и set_documenters должны быть ссылкой на массив!
	# с нормальным массивом работать не будет!
	$aboutdialog->set_authors(\@authors);
	$aboutdialog->set_documenters(\@documenters);
	$aboutdialog->set_website('http://developer.gnome.org');
	$aboutdialog->set_website_label('GNOME Developer Website');

	# мы не хотим показывать заголовок, который по умолчанию был бы
	# 'About AboutDialog Example'
```

```
        # мы должны сбросить заголовок  messagedialog после определения
        # названия программы
        $aboutdialog->set_title('');

        # для закрытия aboutdialog когда кнопка 'close' нажата
        # посылается сигнал 'response' который обрабатывается функцией on_close
        $aboutdialog->signal_connect('response'=>\&on_close);
        # показать
        $aboutdialog->show();
        }

# уничтожить
sub on_close {
        my ($aboutdialog) = @_;
        $aboutdialog->destroy();
        }
```

используемые метоы виджета AboutDialog

В строке 41 сигнал *'activate'* вызывает функцию *about_cb()* используя *$widget->signal_connect("signal" => \&callback function)*. См. сигналы и функции для более детального понимания.

10. 3. Диалог на примере вывода в консоль(Message Dialog)



Вывод в консоль реакций на нажатия клавишь для дальнейшей работы.

Код для того примера

```
#!/usr/bin/perl

use strict;
use Gtk3 -init;
use Glib ('TRUE','FALSE');

my $window = Gtk3::Window->new('toplevel');
$window->set_title('MessageDialog Example');
$window->set_default_size(400,200);
$window->signal_connect('destroy'=>sub {Gtk3->main_quit;});

my $label = Gtk3::Label->new();
$label->set_text('This appliccation goes boom');
```

```
# выкидное меню создается в методе create_menubar (см. ниже)
my $menubar = create_menubar();

# выкидное меню добавляется на виджет лейбла в виджете vertical box
my $vbox = Gtk3::Box->new('vertical', 0);

# Вместо спользования сетки для поицонирования можно виджеты располагать
# последовательно с опциями Expand и Fill установленными в FALSE или FALSE!
$vbox->pack_start($menubar,FALSE,FALSE,0);
$vbox->pack_start($label,TRUE,TRUE,0);

$window->add($vbox);

$window->show_all();
Gtk3->main();

sub create_menubar {
        # содаем выкидное меню
        my $menubar=Gtk3::MenuBar->new();

        # create a menubar itemсоздаем его  элемент
        my $menubar_item=Gtk3::MenuItem->new('Anwendung');

        # добавляем его в менюбар
        $menubar->insert($menubar_item,0);

        # содаем меню
        my $menu=Gtk3::Menu->new();

        # добавляем дв элемента в меню
        my $item1=Gtk3::MenuItem->new('Message');
        $item1->signal_connect('activate'=>\&message_cb);

        my $item2=Gtk3::MenuItem->new('Quit');
        $item2->signal_connect('activate'=> sub {Gtk3->main_quit();});

        $menu->insert($item1,0);
        $menu->insert($item2,1);

        # добавляем меню в выкидное меню(не элемент!!)
        $menubar_item->set_submenu($menu);

        # возвращаем собранное  выкидное меню
        return $menubar;
        }

# вызываем функцию 'activate' из message_action
# в меню основного окна
sub message_cb {
        #  Gtk3::MessageDialog
        # пааметры такие (parent,flags,MessageType,ButtonType,message)
        my $messagedialog = Gtk3::MessageDialog->new($window,
```

```
                                                'modal',
                                                'warning',
                                                'ok_cancel',
                                                'This action will cause the universe to stop
existing.');
        # обрабатываем нажатие кнопки далога
        # функцией dialog_response
        $messagedialog->signal_connect('response'=>\&dialog_response);
        # показываем messagedialog
        $messagedialog->show();
        }

sub dialog_response {
        my ($widget, $response_id) = @_;

        # если нажали  OK (-5)
        if ($response_id eq 'ok') {
                print "boom \n";
                }

        # если нажали CANCEL (-6)
        elsif ($response_id eq 'cancel') {
                print "good choice \n";
                }

        # если уничтожили диалог ( нажатием кнопки ESC)
        elsif ($response_id eq 'delete-event') {
                print "dialog closed or cancelled \n";
                }

        # закрываем все
        $widget->destroy();
        }
```

Useful methods for a MessageDialog widget

В строке 45 сигнал *'activate'* вызывает функцию *message_cb()* используя *$widget->signal_connect("signal" => \&callback function)*. См. сигналы и функции для более детального понимания.

- В конструкторе MessageDialog необходимо выставить флаги *"destroy_with_parent"* (если было закрыто окно, породившее окно мессаджиалога) или *"modal"* (никаких действий с основной программой пока не вышли из диалога).
- В конструкторе MessageDialog необходимо установить что делать: *"info"*, *"warning"*, *"question"*, *"error"*, *"other"* зависит от того что будет делать программа.
- В конструкторе MessageDialog кнопки должны быть следующих типов *"none"*, *"ok"*, *"close"*, *"cancel"*, *"yes_no"*, *"ok_cancel"*, или иного типа, определенного в *add_button()* в *Gtk3::Dialog*.

- Можно в MessageDialog указывать и картинки

```
$image = Gtk3::Image->new();
$image->set_from_stock("caps_lock_warning", "dialog");
$image->show();
$messagedialog->set_image($image);
```

где "caps_lock_warning" одно из зареервированных Stock Items. Также можно использовать и виджет Image для замены или добавления текста к элементам диалога например так *$image->set_from_file('filename.png')*.

- *format_secondary_text('some secondary message')* устанавливает формат дочернего сообщения, отцовское выводит шрифт в формате *bold*.

11. Меню, панели инструментов и подсказки (использование Glade и GtkBuilder)

NOTE: Для эффективного использования программ с использованием панелей инструментов и меню вы должны содать ваше программу как Gtk3::Application в не "main-loop"-style. Следовательно вам нужно импортировать **Glib::IO**, которое позволит вам использовать воможности GMenus, GActions, GApplications, Gtk3::Applications, Gtk3::ApplicationWindows etc.pp.. Основные инструкции для утановки расширений Glib::IO и содания Gtk3::Application или Gtk3::ApplicationWindows вы можете найти в части "1. 2 ApplicationWindows".

Несмотря на то, что реализация Glib::IO находится в очень ранней стадии разработки, создание Меню или Действий(Menus and Actions) кажись, хорошо работает в данный момент. Если вы, однако, хотите обойтись бе использования Glib::IO, вы можете найти примеры здесь here

Но усовершенствованые методы, как например создание ваших меню с помощью xml-файлов ии с помощью Glade, будут довольно затруднительны или вообще не будут работать бе некоторых классов (напрмер Gtk3::UIManager).

11. 1. Меню(GMenu)

GtkApplication с простейшим GMenu и SimpleActions.

Код

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))
# Это необходимо, главным образом, для Gtk3::Application и еще для кое-чего
# более ранняя реалиация лежит тут
#  https://git.gnome.org/browse/perl-Glib-IO (и не опубликована на CPAN!)
# Будем надеяться, что этот модуль упрощает использование Gio API в будущем
# (см. также примечания выше).

BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}


# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое

package MyWindow;
use strict;
use warnings;
use Gtk3;
use Glib qw/TRUE FALSE/;
# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow

use base 'Gtk3::ApplicationWindow';

 sub new {
          my ($window, $app) = @_;
          $window = bless Gtk3::ApplicationWindow->new($app);
          $window->set_title ('GMenu Example');
          $window->signal_connect( 'delete_event' => sub {$app->quit()} );

          return $window;
}


# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
# испускает Gtk3::Application.
# смотри ниже
```

```
package main;
use strict;
use warnings;

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.id', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;


# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
        my ($app) = @_;

        # создаем меню
        my $menu = Glib::IO::Menu->new();

        # добавляем к меню три опции
        $menu->append('New', 'app.new');
        $menu->append('About','app.about');
        $menu->append('Quit','app.quit');

        # установим меню как меню приложения
        $app->set_app_menu($menu);

        # добавляем действие для опции меню "new"
        my $new_action = Glib::IO::SimpleAction→new('new',undef);

        # вызываем обработчик действия new_cb
        $new_action->signal_connect('activate'=>\&new_cb);
        # добавляем действие к приложению
        $app->add_action($new_action);

        # опция "about"
        my $about_action = Glib::IO::SimpleAction->new('about',undef);
        $about_action->signal_connect('activate'=>\&about_cb);
        $app->add_action($about_action);

        # опция "quit"
        my $quit_action = Glib::IO::SimpleAction->new('quit',undef);
        $quit_action->signal_connect('activate'=>\&quit_cb);
```

```
        $app->add_action($quit_action);
}

sub _build_ui {
        my ($app) = @_;
        my $window = MyWindow->new($app);
        $window->show();
}

sub _cleanup {
        my ($app) = @_;
}

# вызываем функцию для "new"
sub new_cb {
        print "This does nothing. It is only a demonstration. \n";
}

# вызываем функцию для "about"
sub about_cb {
        print "No AboutDialog for you. This is only a demonstration \n";
}

# вызываем функцию для "quit"
sub quit_cb {
        print "You have quit \n";
        $app->quit();
}
```

Используемые метоы для виджета Menu (To DO!)

В строке 76 сигналl *'activate'* от действия *new_action* (не меню!) вызывает функцию *new_cb()* используя *$new_action->signal_connect('signal'=>\&callback function)*. См. сигналы и функции для более детального понимания.

Испольуемые методы для GSimpleAction:

- Для создания нового действия, которое находится в состояиии, которое является не сохраняющим состояние, т.е. действие, которое не сохраняет или зависит от состояния, данного самим действием, используют

```
my $action = Glib::IO::SimpleAction->new('name', parameter_type)
```

  где *'name'* имя действия и *parameter_type* типа параметра которое вызывает действие при активации. Оно может быть *undef*, или *Glib::VariantType->new('s')* если тип параметра является *str*, или символом, который определен здесь. Для создания дейтвия, не имеющего фиксированного состояния, используем:

```
my $action = Glib::IO::SimpleAction->new_stateful('name', parameter_type, initial_state)
```

где переменная *initial_state* определена как GVariant - к примеру Glib::Variant->new_string('start'); список вариантов определен здесь.

- *set_enabled(TRUE)* включает действие; действие должно быть активированно, чтобы быть измененным внешними вызывающими сторонами. Эти изменения должен делать только конструктор. Пользоваелю не следует вмешиваться в эти действия.
- *set_state(state)*, где *state* определена как GVariant, устанавливает состояние действия, обновляя переменную 'state' заданным значением. Эти изменения должен делать только конструктор. Пользователи должны для этого вызывать change_state(state) для запашиваемого изменения.

Методы GMenu:

- для добавления элемента меню на поцию *position*, испльзуйте функцию *insert(position, 'label', 'detailed_action')*, где *label* - метка, которая появится в меню и *detailed_action* строка, полученная из имени действия, которое мы ожидаем, ызывая тот или иной элемент меню '*app.'*. Более детальное обсуждение этого может быть найдено в главе о строке меню.

  Чтобы добавить или предварительно ожидать элемент в меню используют соответственно *append('label', 'detailed_action')* и *prepend('label', 'detailed_action')*.

- Другой путь добавления элементов в меню – создание их как *GMenuItems* и использование *insert_item(position, $item)*, *append_item($item)*, или *prepend_item($item)*; так, например, можно было бы сделать:

```
my $about = Glib::IO::MenuItem->new('About', 'app.about');
$menu->append_item($about);
```

- Мы можем также добавить целый подраздел при использовании меню *insert_section(position, 'label', $section)*, *append_section('label', $section)*, or *prepend_section('label', $section)*, где *label* заголовок подменю.
- Для добавления подменю, можно испольовать функцию *insert_submenu(position, 'label', $section)*, *append_submenu('label', $section)*, или *prepend_submenu('label', $section)*, где *label* заголовок подменю.
- Чтобы удалить элемент из меню, использовать *remove(position)*.
- Для установки названия меню, используйте *set_label('label')*.

11. 2. виджет MenuButton

Виджет GtkMenuButton используется для показа меню при его активации. Это меню может быть представлено как GtkMenu, или какабстрактное GmenuModel. Виет GtkMenuButton может содержать любой допустимый дочерний виджет . Т.е. он может содержать любой стандартный GtkWidget. Обычно используемый потомок описывается в GtkArrow.

Code used to generate this example

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))
# Это необходимо, главным образом, для Gtk3::Application и еще для кое-чего
# более ранняя реалиация лежит тут
#  https://git.gnome.org/browse/perl-Glib-IO (и не опубликована на CPAN!)
# Будем надеяться, что этот модуль упрощает использование Gio API в будущем
# (см. также примечания выше).
BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}


# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое

package MyWindow;
use strict;
use warnings;
use Gtk3;
use Glib ('TRUE', 'FALSE');
```

```
# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow
use base 'Gtk3::ApplicationWindow';

sub new {
	my ($window, $app) = @_;
	$window = bless Gtk3::ApplicationWindow->new($app);
	$window->set_title ('MenuButton Example');
	$window->set_default_size(600,400);
	$window->signal_connect( 'delete_event' => sub {$app->quit()} );

	my $grid = Gtk3::Grid->new();

	# кнопка меню
	my $menubutton = Gtk3::MenuButton->new();
	$menubutton->set_size_request(80,35);

	$grid->attach($menubutton, 0, 0, 1, 1);

	# меню с двумя действиями
	my $menumodel = Glib::IO::Menu->new();
	$menumodel->append('New', 'app.new');
	$menumodel->append('About','win.about');

	# подменю с одним действием
	my $submenu = Glib::IO::Menu->new();
	$submenu->append('Quit','app.quit');
	$menumodel->append_submenu('Other',$submenu);

	# создаем подменю
	$menubutton->set_menu_model($menumodel);

	# некоторые действия для меню
	my $about_action = Glib::IO::SimpleAction->new('about',undef);
	$about_action->signal_connect('activate'=>\&about_cb);
	$window->add_action($about_action);

	$window->add($grid);

	return $window;
}

# вызов функции для "about"
sub about_cb {
	print "No AboutDialog for you. This is only a demonstration \n";
}

# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
# испускает Gtk3::Application.
# смотри ниже
```

```
package main;
use strict;
use warnings;

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;

# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
	my ($app) = @_;

	# действия приложения
	my $new_action = Glib::IO::SimpleAction->new('new',undef);
	$new_action->signal_connect('activate'=>\&new_cb);
	$app->add_action($new_action);

	my $quit_action = Glib::IO::SimpleAction->new('quit',undef);
	$quit_action->signal_connect('activate'=>\&quit_cb);
	$app->add_action($quit_action);
}

sub _build_ui {
	my ($app) = @_;
	my $window = MyWindow->new($app);
	$window->show_all();
}

sub _cleanup {
	my ($app) = @_;
}

# вызов функции для действия "new"
sub new_cb {
	print "This does nothing. It is only a demonstration. \n";
}

# вызов функции для действия "quit"
sub quit_cb {
	print "You have quit \n";
```

```
        $app->quit();
}
```

Используемые методы для виджета MenuButton

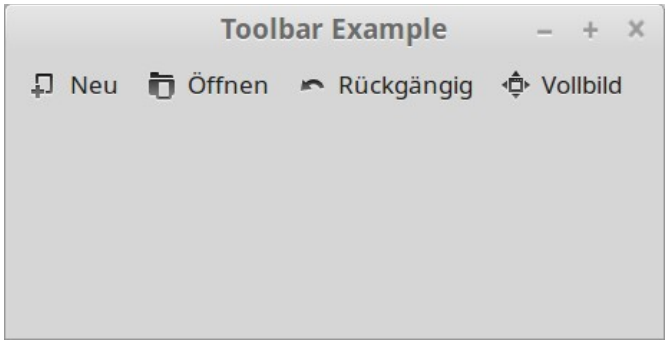
В строке 57 сигнал *'activate'* для действия *about_action* выывет функцию *about_callback()* using *$action->signal_connect('signal'=>\&callback function)*. См. сигналы и функции для более детального понимания.

Позиционирование меню определяется свойством *'direction'* кнопки меню и свойствами *'halign'* или *'valign'* свойств меню. Для примера если направление *"down"* (другая опция: *"up"*) и горизонтальное выравнивание*"start"* (другие опции: *"center"* and *"end"*), меню будет расположено ниже кнопки, со стартовым краем (в зависимости от текстового направления) меню выровненным относительно края кнопки основного меню. Если недостаточно пространства ниже стартовой кнопки, меню показывается выше кнопки.

В части *vertical alignment*, возможные ArrowType *directions* могут быть *"left"* и *"right"* а также *vertical alignment* могут быть *"start"*, *"center"* или *"end"*.

set_align_widget(alignment) и *set_direction(direction)* могут использовать эти свойства.

11. 3. Панель инструментов(Toolbar)



Пример панели инструментов с кнопками и иконками.

Код

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))
# Это необходимо, главным образом, для Gtk3::Application и еще для кое-чего
# более ранняя реалиация лежит тут
#  https://git.gnome.org/browse/perl-Glib-IO (и не опубликована на CPAN!)
# Будем надеяться, что этот модуль упрощает использование Gio API в будущем
# (см. также примечания выше).
```

```
BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}

# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое

package MyWindow;

use strict;
use warnings;

# установить прагму чтобы крректно отображать символы типа умляутов
# или специфических для других яыков символов типа буквы ё или й

use utf8;

# установить "Line Discipline" для станартного вывода в UTF 8 моду.  Таким
# образом, терминал не пытается преобразовать строку снова в LATIN-1

binmode STDOUT, ':utf8';

use Gtk3;
use Glib ('TRUE', 'FALSE');

# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow

use base 'Gtk3::ApplicationWindow';

sub new {
        my ($window, $app) = @_;
        $window = bless Gtk3::ApplicationWindow->new($app);
        $window->set_title ('Toolbar Example');
        $window->set_default_size(400,200);
        $window->signal_connect( 'delete_event' => sub {$app->quit()} );

        # ставим сетку
        my $grid = Gtk3::Grid->new();

        # создаем панель инструментов
        #
        my $toolbar = Gtk3::Toolbar->new();

        # с горизонтальным промежутком
        $toolbar->set_hexpand(TRUE);

         # то, которое является основной панелью инструментов
         # приложения - кажется, не работает в perl
```

```
        $toolbar->get_style_context()->add_class('Gtk3::STYLE_CLASS_PRIMARY_TOOLBAR'
);

        # создать кнопку с иконкой для действия "new"
        my $new_icon = Gtk3::Image->new_from_icon_name('document-new', '16');
        my $new_button = Gtk3::ToolButton->new($new_icon,'Neu');
        # показать лейбл
        $new_button->set_is_important(TRUE);
        # поставить кнопку на позицию 0 панели инструментов
        $toolbar->insert($new_button,0);
        # показать кнопку
        $new_button->show();
        # связать название имени действия с кнопкой.
        # The action controls the application ($app)
        $new_button->set_action_name('app.new');

        # создать кнопку с иконкой для действия "open"
        my $open_icon = Gtk3::Image->new_from_icon_name('document-open', '16');
        my $open_button = Gtk3::ToolButton->new($open_icon,'Öffnen');
        # показать лейбл
        $open_button->set_is_important(TRUE);
        # поставить кнопку на позицию 1 панели инструментов
        $toolbar->insert($open_button,1);
        # показать кнопку
        $open_button->show();
        $open_button->set_action_name('app.open');

        # создать кнопку с иконкой для действия "undo"
        my $undo_icon = Gtk3::Image->new_from_icon_name('edit-undo', '16');
        my $undo_button = Gtk3::ToolButton->new($undo_icon,'Rückgängig');
        # показать лейбл
        $undo_button->set_is_important(TRUE);
        # поставить кнопку на позицию 2 панели инструментов
        $toolbar->insert($undo_button,2);
        # показать кнопку
        $undo_button->show();
        $undo_button->set_action_name('win.undo');

        # создать кнопку с иконкой для действия  'fullscreen'
        my $fullscreen_icon = Gtk3::Image->new_from_icon_name('view-fullscreen', '16');
        my $fullscreen_button = Gtk3::ToolButton->new($fullscreen_icon,'Vollbild');
        # показать лейбл
        $fullscreen_button->set_is_important(TRUE);
        # поставить кнопку на позицию 3 панели инструментов
        $toolbar->insert($fullscreen_button,3);
        # показать кнопку
        $fullscreen_button->show();
        $fullscreen_button->set_action_name('win.fullscreen');

        # показать панель инструментов
        $toolbar->show();
```

```
	# добавить панель инструментов на сетку
	$grid->attach($toolbar, 0, 0, 1, 1);

	# обавить сетку на окно
	$window->add($grid);

	# создать действия для контроля окна и обработки сигналов
	# (смотри ниже)

	# undo
	my $undo_action = Glib::IO::SimpleAction->new('undo',undef);
	$undo_action->signal_connect('activate'=>\&undo_callback);
	$window->add_action($undo_action);

	# fullscreen
	my $fullscreen_action = Glib::IO::SimpleAction->new('fullscreen',undef);
	 # мы должны передать виджету панели инстументов где случился
	 # сигнал, и базовой панели вызов функций как анонимный массив ссылок

	$fullscreen_action->signal_connect('activate'=>\&fullscreen_callback, [$fullscreen_button,
$window]);
	$window->add_action($fullscreen_action);

	return $window;
}

sub fullscreen_callback {
	# IMPORTANT: the second argument that is given is $paramter which here is undef!!!
	my ($action, $parameter, $ref) = @_;
	my $toolbutton = $$ref[0];
	my $window = $$ref[1];
	# receive the GDK Window of the MyWindow object
	my $gdk_win = $window->get_window();
	# взять состояние флагов  GDK Window
	my $is_fullscreen = $gdk_win->get_state();
	# Проверьте, установлен ли полноэкранный флаг
	if ($is_fullscreen =~ m/fullscreen/) {
		$window->unfullscreen();
		my $fullscreen_icon = Gtk3::Image->new_from_icon_name('view-fullscreen', '16');
		$toolbutton->set_icon_widget($fullscreen_icon);
		$toolbutton->set_label('Vollbild');
		$fullscreen_icon->show();
	}
	else {
		$window->fullscreen();
		my $leave_fullscreen_icon = Gtk3::Image->new_from_icon_name('view-restore',
'16');
		$toolbutton->set_icon_widget($leave_fullscreen_icon);
		$toolbutton->set_label('Vollbild verlassen');
		$leave_fullscreen_icon->show();
	}
}
```

```
sub undo_callback {
	my ($widget) = @_;
	print "You clicked \"Rückgängig\" \n";
	}


# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
# испускает Gtk3::Application.
# смотри ниже

package main;

use strict;
use warnings;

# установить прагму чтобы корректно отображать символы типа умляутов
# или специфических для других яыков символов типа буквы ё или й

use utf8;

# установить "Line Discipline" для станартного вывода в UTF 8 моду.  Таким
# образом, терминал не пытается преобразовать строку снова в LATIN-1

binmode STDOUT, ':utf8';

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;


# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
	my ($app) = @_;

	# создать действия, которые контролируют поведение окна и
	# связать их сигналы с соответствующими функцими
```

```
        # new
        my $new_action = Glib::IO::SimpleAction->new('new',undef);
        $new_action->signal_connect('activate'=>\&new_callback);
        $app->add_action($new_action);

        # open
        my $open_action = Glib::IO::SimpleAction->new('open',undef);
        $open_action->signal_connect('activate'=>\&open_callback);
        $app->add_action($open_action);
}

sub _build_ui {
        my ($app) = @_;
        my $window = MyWindow->new($app);
        $window->show_all();
}

sub _cleanup {
        my ($app) = @_;
}

# вызов функции для "new"
sub new_callback {
        print "You clicked \"Neu\". \n";
}

# вызов функции для "open"
sub open_callback {
        print "You clicked \"Öffnen\" \n";
}
```

Используемые методы для виджета панели интструментов Toolbar

В строке 114 сгнал *'activate'* от события *undo_action* вызывает функцию *about_callback()* using *$action->signal_connect('signal'=>\&callback function)*. См. сигналы и функции для более детального понимания.

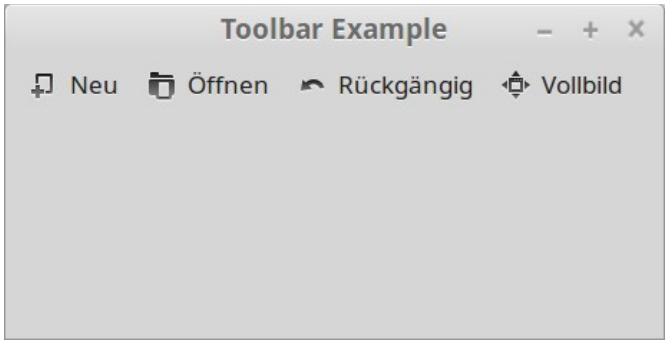
- Используйте *insert($tool_item, position)* для вставки the *tool_item* на позицию *position*. если *position* отрицательна, элемент добавляется в конец виджета панели инструментов.
- *get_item_index($tool_item)* вовращает позицию tool_item на панели инструментов.
- *get_n_items()* вовращает число элементов в панели инструментов; *get_nth_item(position)* возващает позицию элемента.
- Если панель инструментов не умещается на экране, и фенкция *set_show_arrow(TRUE)* установлена в значение TRUE то показывается стрелка переполнения меню.
- *set_icon_size(icon_size)* установить размер иконок панели инструментов; *icon_size*

могут принимать оно из нескольких значений: *"invalid"*, *"menu"*, *"small_toolbar"*, *"large_toolbar"*, *"button"*, *"dnd"*, *"dialog"*. Это должно использоваться только для панелей инструментов специального назначения, нормальные панели инструментов приложения должны уважать пользовательские настройки за размер значков. *unset_icon_size()* сбрасывает предпочительные настройки с *set_icon_size(icon_size)*, позволяя задать необходимые настройки размеа иконок.

- *set_style(style)*, где стиль один из следующих *"icons"*, *"text"*, *"both"*, *"both_horiz"*, задает отображение панели инструментов : только иконки, только текст, или вместе(верикально или рядом друг с другом). Для изменения испольуйте функцию *unset_style()*.

11. 4. Всплываюие подсказки(Tooltips (to do))

11. 5. Виджет панели инструментов, создаваемый с помощью Glade (Toolbar created using Glade)

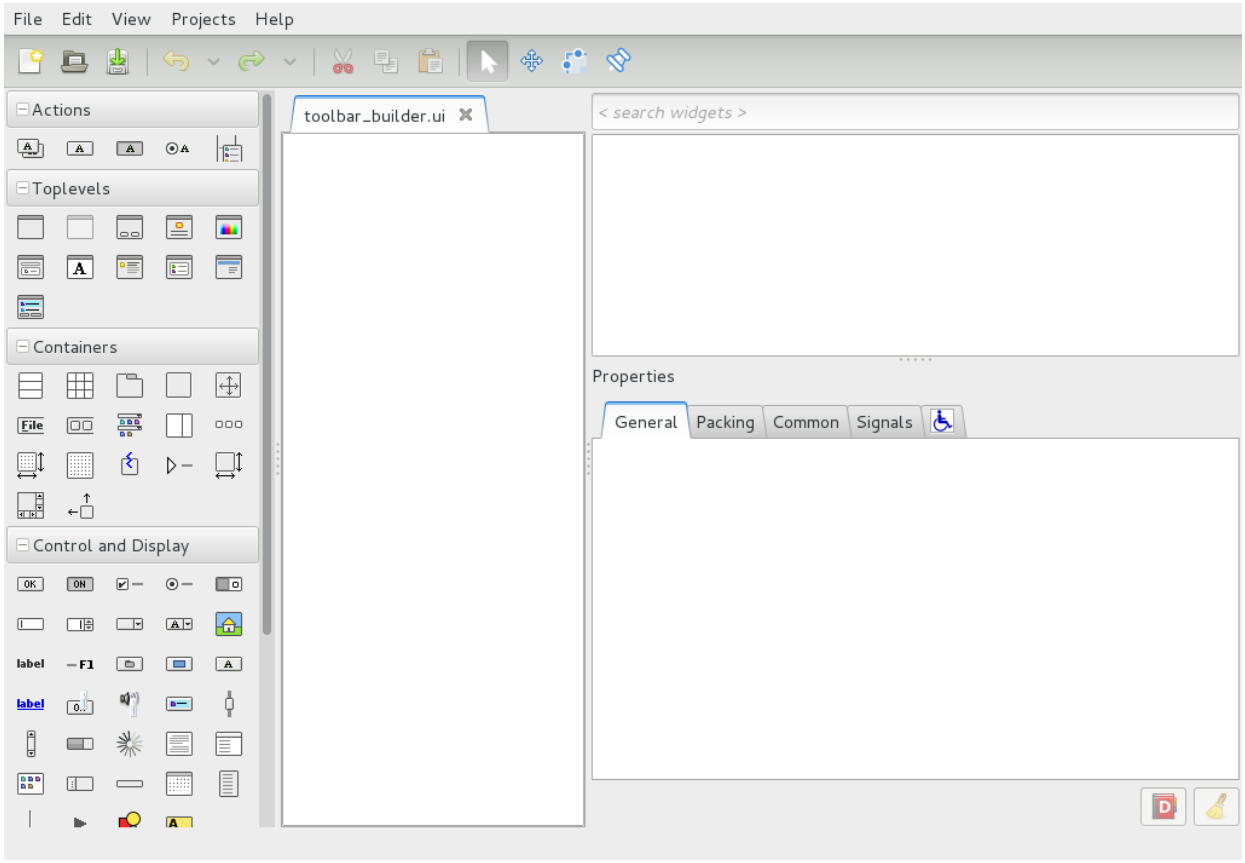


Этот пример подобен примеру панели инструментов выше, кроме того, что для её создания используется Glade и XML .ui файл

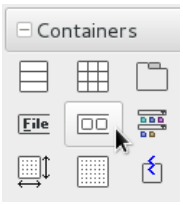
Создание панели инструментов при помои Glade

Испольуем Glade Interface Designer:

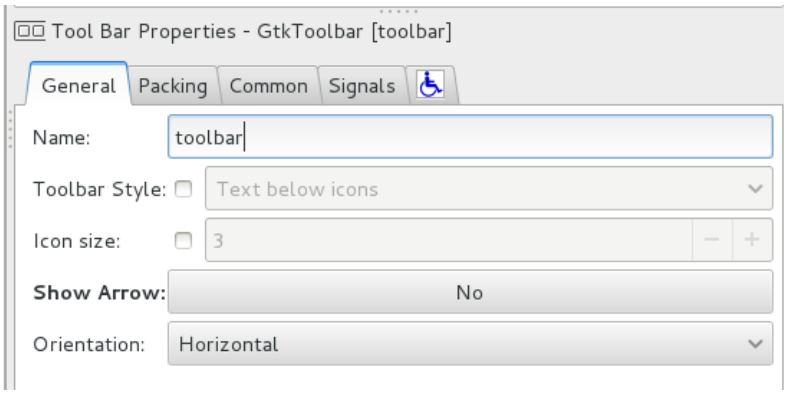
1. открываем Glade и сохраняем файл в отдельной директории как 11_5_*toolbar_builder.ui*



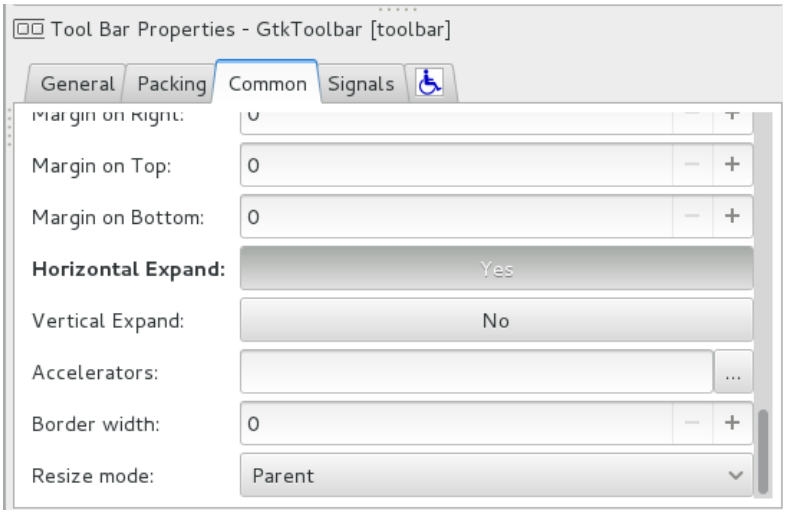
2. На панели Containers слева правой кнопкой мышки выбираем иконку панели инструментов добавляем её нажав на Add widget в панель по центру.



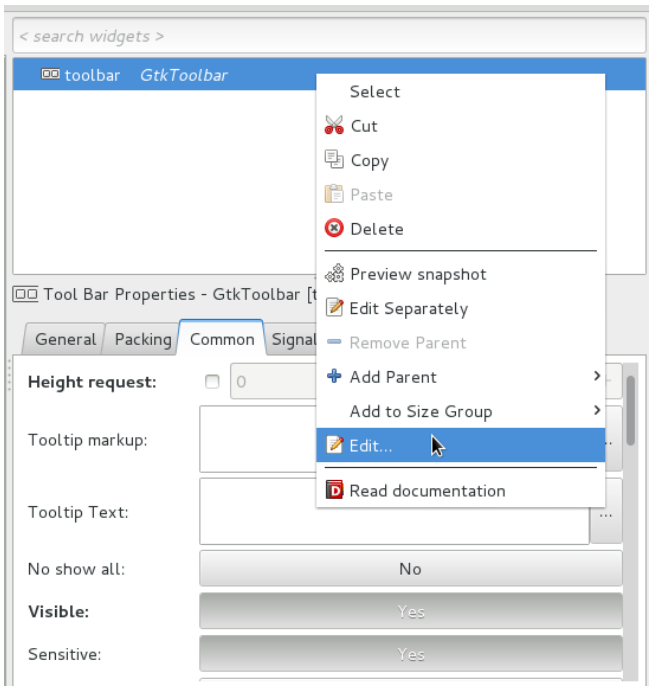
3. Во вкладкке General снизу справа меняем имя Name и Show Arrow в положение No.



4. во вкладке Common, устанавливаем Horizontal Expand в Yes.



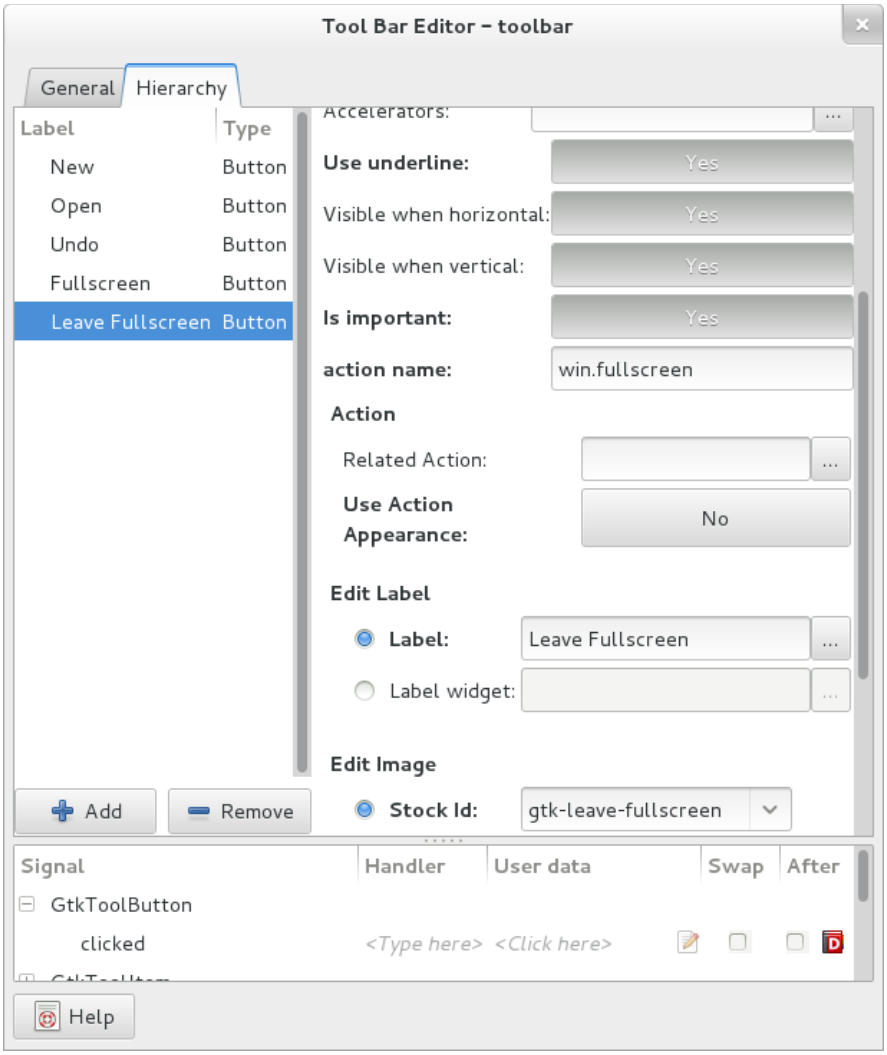
5. Правой кнопкой мышки выбираем в свойствах виджета Edit. Откроется Tool Bar Editor



6. Мы хотим добавить 5 кнопок ToolButtons: New, Open, Undo, Fullscreen и Leave Fullscreen. Сначала, надо добавить New ToolButton.
    1. Кликнуть в Hierarchy tab, нажать Add.
    2. Изменить имя ToolItem на new_button.
    3. Прокрутив ниже установить Is important в положение Yes. This will cause the label of the ToolButton to be shown, when you view the toolbar.
    4. Ввести название действия(Action New): app.new.
    5. Изменить название лейбла кнопки на New.
    6. Выбрать иконку на лейбл Select the New Stock Id з выпадающего меню или ввести навание иконки gtk-new.

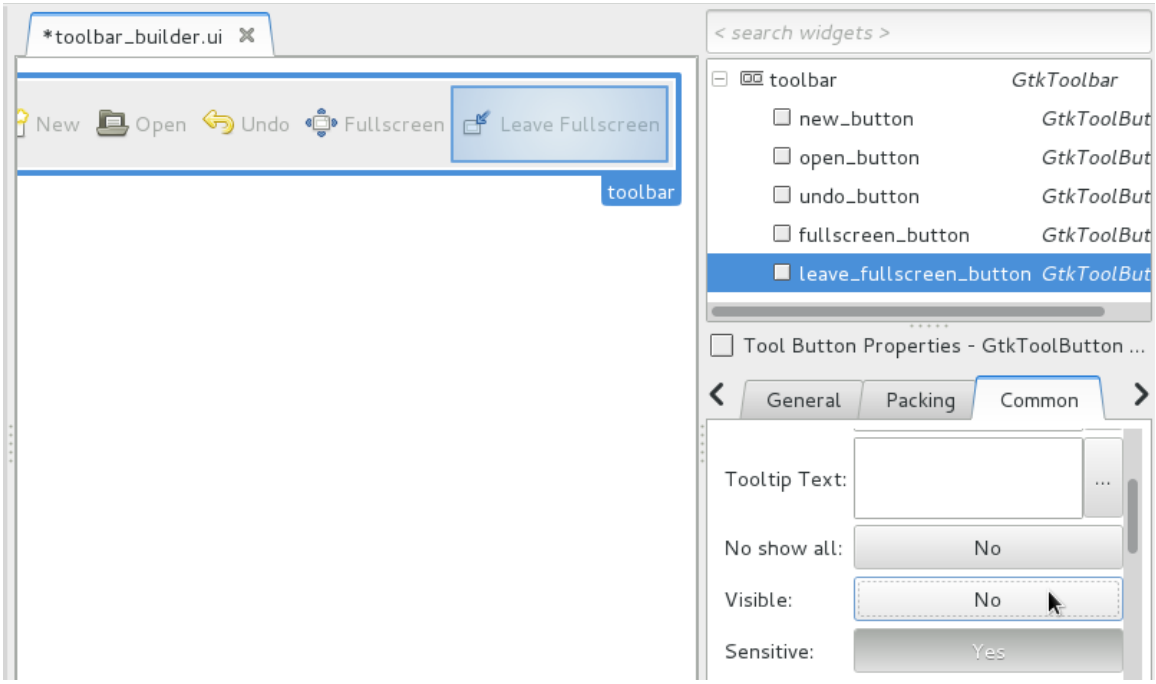
Повторить предыдущие шаги для остальных кнопок согласно таблице:

Name	Is important	Action name	Label	Stock Id
open_button	Yes	app.open	Open	gtk-open
undo_button	Yes	win.undo	Undo	gtk-undo
fullscreen_button	Yes	win.fullscreen	Fullscreen	gtk-fullscreen
leave_fullscreen_button	Yes	win.fullscreen	Leave Fullscreen	Gtk-leave-fullscreen



7. Закрыть Tool Bar Editor.

8. Когда ваша программа первый раз запустится , не хотелось бы чтобы Leave Fullscreen ToolButton была видимой до того момента, как ваша программа будет развернута на полный экран. Это можно задать в редакторе интерфейса Glade , установив во вкладке Common tab свойство Visible в положение No. ToolButton будет видимо в интерфейсе разработчика и далее будет вести себя корректно при работе прогаммы. Также заметьте, что метод show_all() может переопределить эту настройку.



9. Сохранитесь и выйдите из Glade

10. Файл XML, созданный Glade приведен ниже. Это описание виджета панели инструментов. На момент написания данного текста опция добавления класса Gtk.STYLE_CLASS_PRIMARY_TOOLBAR в Glade Interface не была реалиована. Можно добавить это в ручную. Чтобы сделать это – добавьте в 9-ю строку XML файла toolbar_builder.ui следующие строчки :

```
    <style>
      <class name="primary-toolbar"/>
    </style>
```

Если Вы их не добавите, то программа , тем не менее, будет работать. Получившаяся панель инструментов будет, однако, выглядеть немного отличающейся от скрншота экрана наверху этой страницы.

Созданный XML файл

```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
 <!-- interface-requires gtk+ 3.0 -->
 <object class="GtkToolbar" id="toolbar">
  <property name="visible">True</property>
  <property name="can_focus">False</property>
  <property name="hexpand">True</property>
  <property name="show_arrow">False</property>
  <child>
   <object class="GtkToolButton" id="new_button">
    <property name="use_action_appearance">False</property>
    <property name="visible">True</property>
    <property name="can_focus">False</property>
    <property name="use_action_appearance">False</property>
```

```
      <property name="is_important">True</property>
      <property name="action_name">app.new</property>
      <property name="label" translatable="yes">New</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-new</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="homogeneous">True</property>
    </packing>
  </child>
  <child>
    <object class="GtkToolButton" id="open_button">
      <property name="use_action_appearance">False</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="use_action_appearance">False</property>
      <property name="is_important">True</property>
      <property name="action_name">app.open</property>
      <property name="label" translatable="yes">Open</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-open</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="homogeneous">True</property>
    </packing>
  </child>
  <child>
    <object class="GtkToolButton" id="undo_button">
      <property name="use_action_appearance">False</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="use_action_appearance">False</property>
      <property name="is_important">True</property>
      <property name="action_name">win.undo</property>
      <property name="label" translatable="yes">Undo</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-undo</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="homogeneous">True</property>
    </packing>
  </child>
  <child>
    <object class="GtkToolButton" id="fullscreen_button">
      <property name="use_action_appearance">False</property>
      <property name="visible">True</property>
      <property name="can_focus">False</property>
      <property name="use_action_appearance">False</property>
      <property name="is_important">True</property>
```

```
      <property name="action_name">win.fullscreen</property>
      <property name="label" translatable="yes">Fullscreen</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-fullscreen</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="homogeneous">True</property>
    </packing>
  </child>
  <child>
    <object class="GtkToolButton" id="leave_fullscreen_button">
      <property name="use_action_appearance">False</property>
      <property name="can_focus">False</property>
      <property name="use_action_appearance">False</property>
      <property name="is_important">True</property>
      <property name="action_name">win.fullscreen</property>
      <property name="label" translatable="yes">Leave Fullscreen</property>
      <property name="use_underline">True</property>
      <property name="stock_id">gtk-leave-fullscreen</property>
    </object>
    <packing>
      <property name="expand">False</property>
      <property name="homogeneous">True</property>
    </packing>
  </child>
 </object>
</interface>
```

Код для этого примера

Напишем код который добавит панель инструментов из созданного нами выше файла:

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))
# Это необходимо, главным образом, для Gtk3::Application и еще для кое-чего
# более ранняя реалиация лежит тут
#  https://git.gnome.org/browse/perl-Glib-IO (и не опубликована на CPAN!)
# Будем надеяться, что этот модуль упрощает использование Gio API в будущем
# (см. также примечания выше).
BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}


# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое
```

```
package MyWindow;

use strict;
use warnings;

# установить прагму чтобы крректно отображать символы типа умляутов
# или специфических для других яыков символов типа буквы ё или й

use utf8;

# установить "Line Discipline" для станартного вывода в UTF 8 моду.  Таким
# образом, терминал не пытается преобразовать строку снова в LATIN-1

binmode STDOUT, ':utf8';

use Gtk3;
use Glib ('TRUE', 'FALSE');

# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow

use base 'Gtk3::ApplicationWindow';

# преведенные переменные будут использованы позднее в методе
# (поэтому мы их объявляем вне нового конструктора
# Дрегие решения:
# 1) Вы можете передавать виджеты как ссылки на анонимные массивы
# в качетве дополнительных параметров функции  fullscreen_cb (см.
# простой пример панели инструментов)
# 2) Вы можете объявить эти переменные как our
# 3) вы можете определить (!) вызов функции в новом конструкторе (не очень хорошо)

my ($fullscreen_button, $leave_fullscreen_button, $window);

sub new {
        $window = $_[0]; my $app = $_[1];
        $window = bless Gtk3::ApplicationWindow->new($app);
        $window->set_title ('Toolbar Example');
        $window->set_default_size(400,200);
        $window->signal_connect( 'delete_event' => sub {$app->quit()} );

        # добавим сетку

        my $grid = Gtk3::Grid->new();

        $window->add($grid);

        # мы должны показать сетку (следовательно и панель инструментов) с show(),
        # как show_all() если хотим, чтобы некоторые картинки в меню
        # присутствовали, но были скрыты,
        #  (подобно leave_fullscreen button)
```

```
	$grid->show();

	# добавляем пользовательский интерфейс(UI), соданныйпри помощи
	# Glade на сетку:
	my $builder = Gtk3::Builder->new();
	# берем файл (ежит в той-же директории)
	$builder->add_from_file('11_5_toolbar_builder.ui') or die 'file not found';
	# добавляем на сетку
	my $toolbar = $builder->get_object('toolbar');
	$grid->attach($toolbar, 0, 0, 1, 1);

	$fullscreen_button = $builder->get_object('fullscreen_button');
	$leave_fullscreen_button = $builder->get_object('leave_fullscreen_button');

	# создать события, которые управляют приложением

	# кнопка undo
	my $undo_action = Glib::IO::SimpleAction->new('undo',undef);
	$undo_action->signal_connect('activate'=>\&undo_callback);
	$window->add_action($undo_action);

	# кнопка fullscreen
	my $fullscreen_action = Glib::IO::SimpleAction->new('fullscreen',undef);
	$fullscreen_action->signal_connect('activate'=>\&fullscreen_callback);
	$window->add_action($fullscreen_action);

	return $window;
}

# вызов для undo
sub undo_callback {
	my ($action, $parameter) = @_;
	print "You clicked \"Undo\" \n";
}

# вызов для fullscreen

sub fullscreen_callback {

	# добавить GDK Window of the MyWindow object
	my $gdk_win = $window->get_window();
	# проверить состояния флагов GDK Window
	my $is_fullscreen = $gdk_win->get_state();
	# проверить флаг полного экрана
	if ($is_fullscreen =~ m/fullscreen/) {
		$window->unfullscreen();
		$leave_fullscreen_button->hide();
		$fullscreen_button->show();
	}
	else {
		$window->fullscreen();
		$fullscreen_button->hide();
```

```
                $leave_fullscreen_button->show();
        }
}


# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
# испускает Gtk3::Application.
# смотри ниже

package main;
use strict;
use warnings;

# установить прагму чтобы корректно отображать символы типа умляутов
# или специфических для других яыков символов типа буквы ё или й

use utf8;

# установить "Line Discipline" для станартного вывода в UTF 8 моду.  Таким
# образом, терминал не пытается преобразовать строку снова в LATIN-1

binmode STDOUT, ':utf8';

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;


# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
        my ($app) = @_;

        # создать действия, которые контролируют поведение окна и
        # связать их сигналы с соответствующими функцими

        # new
        my $new_action = Glib::IO::SimpleAction->new('new',undef);
        $new_action->signal_connect('activate'=>\&new_callback);
        $app->add_action($new_action);
```

```
        # open
        my $open_action = Glib::IO::SimpleAction->new('open',undef);
        $open_action->signal_connect('activate'=>\&open_callback);
        $app->add_action($open_action);
}

sub _build_ui {
        my ($app) = @_;
        my $window = MyWindow->new($app);
        # показать окно - с show() но не show_all() потому что скрытая кнопка
        #  leave_fullscreen_button
        $window->show();
}

sub _cleanup {
        my ($app) = @_;
}

# вызов функции для "new"
sub new_callback {
        print "You clicked \"New\". \n";
}

# вызов функции для "open"
sub open_callback {
        print "You clicked \"Open\" \n";
}
```

Используемые методы для Gtk3::Builder

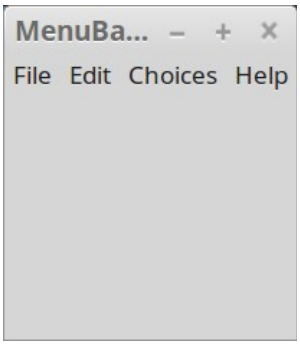
Для наиболее часто используемых методов виджета панели инструментов Toolbar, смотри the Toolbar Chapter.

Gtk3::Builder строит интефейс XML UI .

- *add_from_file('filename')* загружает и парсит файл конфигурации сливает(хи-хи, мержит, как это на русском сказать) его с содержимым Gtk3::Builder.
- *add_from_string('string')* берет строчку и сопосавляе её с потрохами Gtk3::Builder, в результате этого чтонибудь да и выходит.
- *add_objects_from_file('filename', 'object_ids')* часть *add_from_file()*, но загружает только объекты, которые определены в *object_ids* list.
- *add_objects_from_string('string', 'object_ids')* часть *add_from_string()*, но загружает только объекты, которые определены в *object_ids* list.
- *get_object('object_id')* возвращает виджет с определенным *object_id* из загруженных в билдере объектов.
- *get_objects()* возвращает все использованные объекты.

- *connect_signals(handler_object)* связывает сигналы и методы, указанные в *handler_object*. Это могут быть разные объекты, содержащие ключи или аттрибуты , которые вызываются подобно обработчикам сигнала имен в описании нтерфейса, например классы ли хеши(во загнули то!) . В 74 строке сигнал *'activate'* от события *undo_action* вызывает функцию *undo_callback()* используя *$action->signal_connect(signal=>\&callback function)*. См. сигналы и функции для более детального понимания.

11. 6. MenuBar созданный с помощью XML и GtkBuilder (to do)



Виджет панели инструментов, созданный с помощью XML и Gtk3::Builder

Создание паели инструментов(MenuBar) при помощи XML

Для создания всего этого страха добавим еще ужас XML:

1. В своем любимом редакторе содадим файл *11_6_menubar.ui* .
2. Первой строкой в файле введем:

```
<?xml version="1.0"? encoding="UTF-8"?>
```

3. мы хотим создать интерфейс который будет содержать панель инструментов с подменю. Наша панель инструментов будет содержать подменю File, Edit, Choices и Help. Добавим следующий XML код в файл:

```
<?xml version="1.0"? encoding="UTF-8"?>
<interface>
	<menu id="menubar">
		<submenu>
			<attribute name="label">File</attribute>
		</submenu>
		<submenu>
			<attribute name="label">Edit</attribute>
		</submenu>
		<submenu>
			<attribute name="label">Choices</attribute>
		</submenu>
		<submenu>
			<attribute name="label">Help</attribute>
		</submenu>
```

```
        </menu>
    </interface>
```

4. Теперь создадим соответствующий .pl файл и используем Gtk3::Builder для импорта *11_6_menubar.ui* и создания программы

Добавляем панель инструментов на базовую панель(window) с помощью Gtk3::Builder

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))

BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}

# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое

package MyWindow;
use strict;
use warnings;
use Gtk3;
use Glib ('TRUE', 'FALSE');

# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow

use base 'Gtk3::ApplicationWindow';

sub new {
        my ($window, $app) = @_;
        $window = bless Gtk3::ApplicationWindow->new($app);
        $window->set_title ('MenuBar Example');
        $window->set_default_size (200, 200);
        $window->signal_connect( 'delete_event' => sub {$app->quit()} );

        return $window;
}


# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
```

```
# испускает Gtk3::Application.
# смотри ниже

package main;
use strict;
use warnings;

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init);
$app->signal_connect('activate' => \&_build_ui);
$app->signal_connect('shutdown' => sub {$app->quit();});

$app->run(\@ARGV);

exit;

# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
	my ($app) = @_;

	# Добавить UI на сетку
	my $builder = Gtk3::Builder->new();
	$builder->add_from_file('11_6_menubar-firststep.ui') or die 'file not found';

	# используем метод Gtk3::Application->set_menubar('menubar')
	# для добавления к приложению (заметьте НЕ окну!)
	my $menubar=$builder->get_object('menubar');
	$app->set_menubar($menubar);

 }

sub _build_ui {
	my ($app) = @_;
	# соберем Gtk3::ApplicationWindow и его содержимое здесь разделено на классы
	#  ( см. выше )
	my $window = MyWindow->new($app);
	$window->show();
}
```

Запустим программу и увидим картинку, подобную приведенной выше.

Добавление элементов к меню

Добавим два дочерних элемента в элемент File меню: New и Quit. Чтобы это сделать,

добавим в секцию section меню *File* подменю с двумя элементами, файл 11_6_*menubar.ui* теперь должен выглядеть так (строки с 6 по 13 показывают изменение)

menubar.ui

```
<?xml version="1.0"? encoding="UTF-8"?>
<interface>
	<menu id="menubar">
		<submenu>
			<attribute name="label">File</attribute>
			<section>
				<item>
					<attribute name="label">New</attribute>
				</item>
				<item>
					<attribute name="label">Quit</attribute>
				</item>
			</section>
		</submenu>
		<submenu>
			<attribute name="label">Edit</attribute>
		</submenu>
		<submenu>
			<attribute name="label">Choices</attribute>
		</submenu>
		<submenu>
			<attribute name="label">Help</attribute>
		</submenu>
	</menu>
</interface>
```

Подобно этому шаблону можно добавлять любые подменю, например Copy и Paste в Edit, или что-то в About или в Help.

Установка действий

Теперь мы создадим действия для "New" и "Quit", заставив их вызвать соответствующие функции, для примера мы создали “new”:

```
my $new_action = Glib::IO::SimpleAction->new('new',undef);
$new_action->signal_connect('activate'=>\&new_callback);
```

И создадим функцию для обработки 'new':

```
sub new_callback {
	print "You clicked \"New \" \n";
}
```

Тепепрь, в XML файле, соединим элементы меню с действиями посредством добавления аттрибута "action":

```
<item>
	<attribute name="label">New</attribute>
	<attribute name="action">app.new</attribute>
</item>
```

Заметьте, что для действий, связанных с приложением, мы используем префикс *"app."*, но для действий связанных с бовым окном используется префикс *"app."*..

Наконец, в файле perl, мы добавляем действие к приложению или к окну - таким образом, например, *app.new* будет добавлен к приложению в методе _init ()

```
$app->add_action($new_action);
```

См. сигналы и функции для более детального понимания.

Действия: приложение или окно?

Выше, мы создали "new" и "open" действия как часть класса MyApplication. Действия, которые управляют самим приложением, таким как "выход", должны быть созданы так же.

Некоторые действия, такие как "copy" и "paste" имеют дело с окном, а не с приложением. Действия с окнами должны быть созданы как часть класса окна.

Полный пример ниже содержит пример работы как с окном, так и с приложением. Действия с элементами графики обычно включаются в *application menu* . Это не очень хорошая практка, когда действия окна включаются меню приложения. В демонстрационных целях полный пример содержит XML UI, который создает меню приложения, которое включает "New" и "Open" и они взаимодействуют как элементы одного и того же меню.

Выбор подменю и элементов с определенным состоянием

Строки с 30 по 80 включительно содежат конфигурационный файл XML UI для демонтрации использовния XML при создании более сложного меню, чем было описано выше.

Действия, не сохраняющие состояние для подменю Choices не запоминаются, однако есть и такие действия пользователя, которые надо запомнить. Пример:

```
my $shape_action = Glib::IO::SimpleAction->new_stateful('shape', Glib::VariantType->new('s'), Glib::Variant->new_string('line'));
```

где переменные метода таковы: *name*, *parameter type* (в этом смысле, строка - смотри здесь полный лист обозначений действий), *initial state* (в этом смысле, 'line' – булева переменная True должна быть объявлена так *Glib::Variant->new_boolean(TRUE)*, см здесь полный лист обозначений – полная абракадабра, разобраться в контексте и перевести заново)

После создания фиксированного SimpleAction мы вызываем функцию и добавляем работу с графикой(окном) или с приложением, например так:

```
$shape_action->signal_connect('activate'=>\&shape_callback);
$window->add_action($shape_action);
```

Полный XML UI для нашего примера

```
<?xml version="1.0"? encoding="UTF-8"?>
<interface>
	<menu id="menubar">
		<submenu>
			<attribute name="label">File</attribute>
			<section>
				<item>
					<attribute name="label">New</attribute>
					<attribute name="action">app.new</attribute>
				</item>
				<item>
					<attribute name="label">Quit</attribute>
					<attribute name="action">app.quit</attribute>
				</item>
			</section>
		</submenu>
		<submenu>
			<attribute name="label">Edit</attribute>
			<section>
				<item>
					<attribute name="label">Copy</attribute>
					<attribute name="action">win.copy</attribute>
				</item>
				<item>
					<attribute name="label">Paste</attribute>
					<attribute name="action">win.paste</attribute>
				</item>
			</section>
		</submenu>
		<submenu>
```

```
                <attribute name="label">Choices</attribute>
                <submenu>
                <attribute name="label">Shapes</attribute>
                  <section>
                    <item>
                      <attribute name="label">Line</attribute>
                      <attribute name="action">win.shape</attribute>
                      <attribute name="target">line</attribute>
                    </item>
                    <item>
                      <attribute name="label">Triangle</attribute>
                      <attribute name="action">win.shape</attribute>
                      <attribute name="target">triangle</attribute>
                    </item>
                    <item>
                      <attribute name="label">Square</attribute>
                      <attribute name="action">win.shape</attribute>
                      <attribute name="target">square</attribute>
                    </item>
                    <item>
                      <attribute name="label">Polygon</attribute>
                      <attribute name="action">win.shape</attribute>
                      <attribute name="target">polygon</attribute>
                    </item>
                    <item>
                      <attribute name="label">Circle</attribute>
                      <attribute name="action">win.shape</attribute>
                      <attribute name="target">circle</attribute>
                    </item>
                  </section>
            </submenu>
            <section>
                <item>
                  <attribute name="label">On</attribute>
                  <attribute name="action">app.state</attribute>
                  <attribute name="target">on</attribute>
                </item>
                <item>
                  <attribute name="label">Off</attribute>
                  <attribute name="action">app.state</attribute>
                  <attribute name="target">off</attribute>
                </item>
            </section>
            <section>
                <item>
                  <attribute name="label">Awesome</attribute>
                  <attribute name="action">app.awesome</attribute>
                </item>
            </section>
        </submenu>
        <submenu>
                <attribute name="label">Help</attribute>
```

```
                        <section>
                                <item>
                                        <attribute name="label">About</attribute>
                                        <attribute name="action">win.about</attribute>
                                </item>
                        </section>
                </submenu>
        </menu>
        <menu id="appmenu">
                <section>
                        <item>
                                <attribute name="label">New</attribute>
                                <attribute name="action">app.new</attribute>
                    </item>
                    <item>
                                <attribute name="label">Quit</attribute>
                                <attribute name="action">app.quit</attribute>
                    </item>
                </section>
        </menu>
</interface>
```

код

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))
# Это необходимо, главным образом, для Gtk3::Application и еще для кое-чего
# более ранняя реалиация лежит тут
#  https://git.gnome.org/browse/perl-Glib-IO (и не опубликована на CPAN!)
# Будем надеяться, что этот модуль упрощает использование Gio API в будущем
# (см. также примечания выше).

BEGIN {
  use Glib::Object::Introspection;
  Glib::Object::Introspection->setup(
    basename => 'Gio',
    version => '2.0',
    package => 'Glib::IO');
}

# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое

package MyWindow;
use strict;
use warnings;
use Gtk3;
use Glib ('TRUE', 'FALSE');
```

```
# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow

use base 'Gtk3::ApplicationWindow';

sub new {
	my ($window, $app) = @_;
	$window = bless Gtk3::ApplicationWindow->new($app);
	$window->set_title ('MenuBar Example');
	$window->set_default_size (200, 200);
	$window->signal_connect( 'delete_event' => sub {$app->quit()} );

	# действие без создания его начального состояния (name, parameter type)
	my $copy_action = Glib::IO::SimpleAction->new('copy',undef);
	# определяем действию функцию
	$copy_action->signal_connect('activate'=>\&copy_callback);
	# добавляем
	$window->add_action($copy_action);

	# действие без создания его начального состояния (name, parameter type)
	my $paste_action = Glib::IO::SimpleAction->new('paste',undef);
	# определяем действию функцию
	$paste_action->signal_connect('activate'=>\&paste_callback);
	# адобавляем
	$window->add_action($paste_action);

	#  действие с созданием его начального состояния
	#  (name, parameter type, initial state)
	my $shape_action = Glib::IO::SimpleAction->new_stateful('shape',
Glib::VariantType->new('s'), Glib::Variant->new_string('line'));
	# определяем действию функцию
	$shape_action->signal_connect('activate'=>\&shape_callback);
	# добавляем
	$window->add_action($shape_action);

	# действие с созданием его начального состояния
	my $about_action = Glib::IO::SimpleAction->new('about',undef);
	# определяем действию функцию
	$about_action->signal_connect('activate'=>\&about_callback);
	# добавляем
	$window->add_action($about_action);

	return $window;
}

 # для копирования

sub copy_callback {
	print "\"Copy\" activated \n";
}

# для вставки
```

```
sub paste_callback {
	print "\"Paste\" activated \n";
}

# функция для действия с сохранением

sub shape_callback {
	my ($action, $parameter) = @_;
	my $string = $parameter->get_string();
	print "Shape is set to $string \n";
	# Отметьте, что тут состояние сохраняется!
	$action->set_state($parameter);
}

# вызов для About (see AboutDialog example)
sub about_callback {
	my ($action, $parameter) = @_;
	print "About activated \n";
}

# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
# испускает Gtk3::Application.
# смотри ниже

package main;

use strict;
use warnings;

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;

# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
	my ($app) = @_;

	# action without a state created
```

```
        my $new_action = Glib::IO::SimpleAction→new('new',undef);

        # событие вызывает функцию
        $new_action->signal_connect('activate'=>\&new_callback);

        # событие добавляется к приложению
        $app->add_action($new_action);

        # событие без сохранения состояния
        my $quit_action = Glib::IO::SimpleAction->new('quit',undef);

        # событие вызывает функцию
        $quit_action->signal_connect('activate'=>\&quit_callback);
        # событие добавляется к приложению
        $app->add_action($quit_action);

        # событие с сохранением состояния
        my $state_action = Glib::IO::SimpleAction->new_stateful('state',
Glib::VariantType->new('s'), Glib::Variant->new_string('off'));
        # асобытие вызывает функцию
        $state_action->signal_connect('activate'=>\&state_callback);
        # событие добавляется к приложению
        $app->add_action($state_action);

        # событие с сохранением состояния
        my $awesome_action = Glib::IO::SimpleAction->new_stateful('awesome', undef,
Glib::Variant->new_boolean(TRUE));
        # событие вызывает функцию
        $awesome_action->signal_connect('activate'=>\&awesome_callback);
        # событие добавляется к приложению
        $app->add_action($awesome_action);

        # добавляем графический интефейс на сетку
        my $builder = Gtk3::Builder->new();
        $builder->add_from_file('11_6_menubar.ui') or die 'file not found';

        # используем метод Gtk3::Application->set_menubar('menubar')
        # для добалвения менюбара на приложение (Внимание: НЕ на окно!)
        my $menubar=$builder->get_object('menubar');
        $app->set_menubar($menubar);

}

sub _build_ui {
        my ($app) = @_;
        # собираем Gtk3::ApplicationWindow и его содержимое здесь
        # с помощью seperate class (см выше)
        my $window = MyWindow->new($app);
        $window->show();
}

sub _cleanup {
```

```
	my ($app) = @_;
}

# вызов функции для new
sub new_callback {
	print "You clicked \"New \" \n";
}

# вызов функции для  quit
sub quit_callback {
	print "You clicked \"Quit\" \n";
	$app->quit();
}
# вызов функции для state
sub state_callback {
	my ($action, $parameter) = @_;
	my $string = $parameter->get_string();
	print "State is set to $string \n";

	$action->set_state($parameter);
}

# вызов функции для awesome
sub awesome_callback {
	my ($action, $parameter) = @_;
	my $state = $action->get_state();
	my $boolean = $state->get_boolean();
	if ($boolean) {
		print "You unchecked \"Awesome\" \n";
		$action->set_state(Glib::Variant->new_boolean(FALSE));
	}
	else {
		print "You checked \"Awesome\" \n";
		$action->set_state(Glib::Variant->new_boolean(TRUE));
	}
}
```

Мнемонические клавиши и ускорители(Mnemonics and Accelerators)

Лейблы могут содержать мнемонические клавиши. Их можно увидеть как подчеркнутые буквы на заголовочном виджете(labels). Мнемонические клавиши – своеобразные горячие клавиши, которые могут использоваться пользователем для управления графическим приложением с помощью клавиатуры. Для примера напишем "_File" вместо просто "File" в свойствах виджета аголовка в конфигурационном файле 11_6_menubar.ui .

Мнемонические виджеты можно увидеть нажав кнопку Alt на клавиатуре. Нажатие Alt+F откроет меню File.

Так же есть виджеты ускорители, которые позволяют деать некоторые действия при помощи

клавиатуры, если окно приложения находиться в фокусе. Например, очень распространено выйти из приложения, нажав кнопку Ctrl+Q или сохранить файл при помощи For example, it is common to be able to quit an application by pressing Ctrl+Q or to save a file by pressing Ctrl+S. Добавить ускоряющие кнопки можно, выставив аттрибут "accel" для элемента меню.

<attribute name="accel"><Primary>q</attribute> создаст действие Ctrl+Q которое добавляется к виджету заголовка Quit. Ниже, "Primary" ссылается на калвишу Ctrl на компьютере архтектуры PC или ⌘ на Mac.

```
<item>
        <attribute name="label">_Quit</attribute>
        <attribute name="action">app.quit</attribute>
        <attribute name="accel">&lt;Primary&gt;q</attribute>
</item>
```

Автоперевод

Приложения GNOME переведены на многие языки , если вы добавите к метке аттрибут set *translatable="yes"*, то она будет покаана в соответствии с настроенной у Вас локалью:

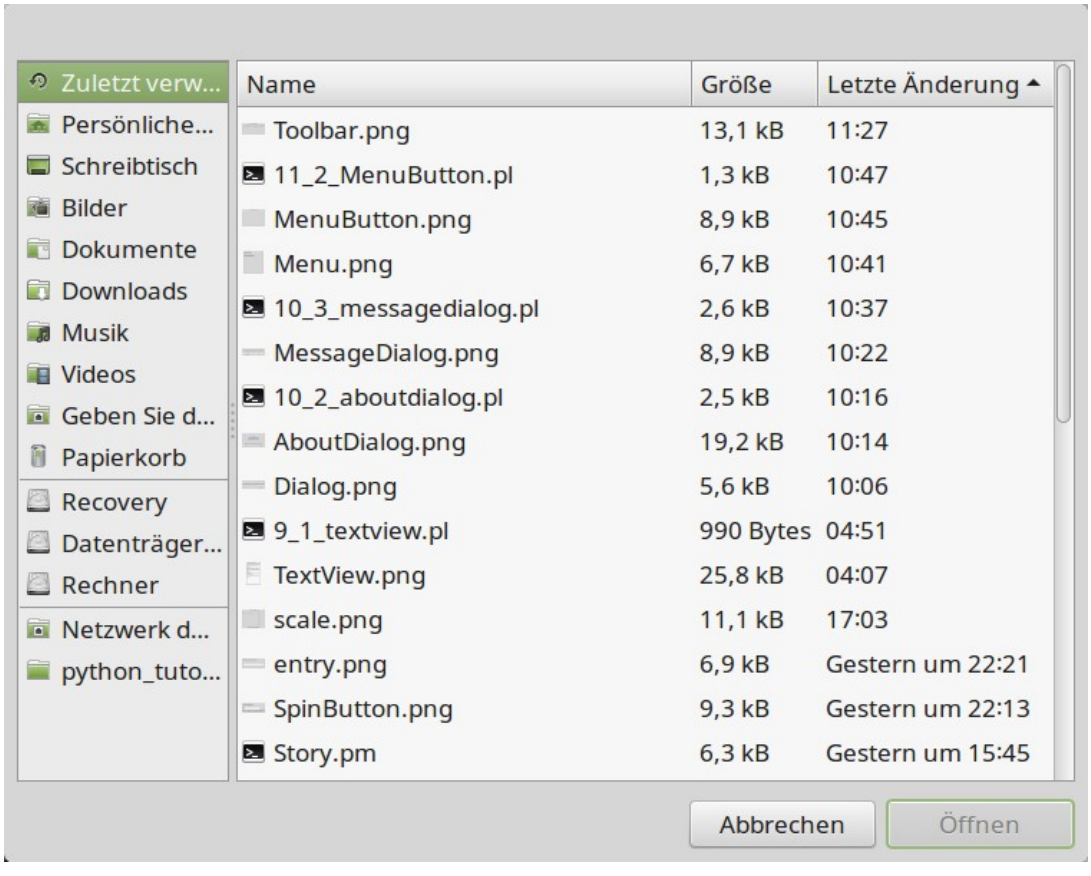
```
<attribute name="label" translatable="yes">Quit</attribute>
```

12. Кнопки выбора (Selectors)

12. 1. ColorButton (to do)

12. 2. FontChooserWidget (to do)

12. 3. Файловый диалог (FileChooserDialog)



Виджет FileChooserDialog сохраняет текстовый документ, который был открыт и/или изменен в виджете TextView.

FileChooserDialog так же можно вызывать при открытии нового документа.

Шаги для создания примера

1. Создайте файл *12_3_filechooser.ui* для описания приложения с элементами "New", "Open", "Save", "Save as", and "Quit". Это можно сделать с помощью виуального редактора интерфейсов Glade, описанного выше . В качестве примера XML файла мы возьмем рассмотренный выше шаблон меню.
2. Создайте Perl программу для Gtk3::TextView с Gtk3::Buffer $buffer, и $file который будет объектом Glib::IO::File и переопределите инициалиацию в *undef*.
3. В этой программе создайте также действия, соответствующие элементам в меню приложения, заставьте действия обрабатываться соответствующими функциями и импортируйте меню в the метод *_init()* с помощью *Gtk3::Builder*.
4. Действия "New" и "Quit" достаточно хорошо описаны в коде, который иллюстрирует работу виджета FileChooserDialog. См. сигналы и функции для более детального понимания. .
5. Вызов "Open" создает и открывает объект Gtk3::FileChooserDialog для действия "Open", который вступает во взаимоействие с другой обрабатывающей функцией для любой из кнопок "Open" или "Cancel" виджета FileChooserDialog.
6. Действие "Save as" работает подобно действию "Open", но вызов функции, обрабатывающей событие "Save" зависит от некоторых особенностей функции *save_to_file()*.
7. Кнопка "Save" может работать и в случае сохранения нового файла, тогда она должна возвращать действие "Save as"; чтобы ввести имя файла, это определяется функцией *save_to_file()*.
8. В заключении, *save_to_file()*: см. строчки в примере 230 - 267.

XML file which creates the app-menu

```
<?xml version="1.0"?>
<interface>
  <menu id="appmenu">
    <section>
      <item>
        <attribute name="label">New</attribute>
        <attribute name="action">win.new</attribute>
      </item>
```

```
    <item>
      <attribute name="label">Open</attribute>
      <attribute name="action">win.open</attribute>
    </item>
  </section>
  <section>
    <item>
      <attribute name="label">Save</attribute>
      <attribute name="action">win.save</attribute>
    </item>
    <item>
      <attribute name="label">Save As...</attribute>
      <attribute name="action">win.save-as</attribute>
    </item>
  </section>
  <section>
    <item>
      <attribute name="label">Quit</attribute>
      <attribute name="action">app.quit</attribute>
    </item>
  </section>
 </menu>
</interface>
```

Code used to generate this example

```
#! /usr/bin/perl

# Забиндите(сделайте привяку) к Gio API в Perl программу (just copy&paste ;-))
# Это необходимо, главным образом, для Gtk3::Application и еще для кое-чего
# более ранняя реалиация лежит тут
#  https://git.gnome.org/browse/perl-Glib-IO (и не опубликована на CPAN!)
# Будем надеяться, что этот модуль упрощает использование Gio API в будущем
# (см. также примечания выше).

BEGIN {
 use Glib::Object::Introspection;
 Glib::Object::Introspection->setup(
   basename => 'Gio',
   version => '2.0',
   package => 'Glib::IO');
}



# CLASS MyWindow, который мы создали в Gtk3::ApplicationWindow и его содержимое

package MyWindow;
use strict;
use warnings;
use utf8;
use Gtk3;
use Glib ('TRUE', 'FALSE');
use Encode;
```

```
# Наш класс должен быть подклассом Gtk3::ApplicationWindow для наследования
# методов и свойств etc.pp. Gtk3::ApplicationWindow

use base 'Gtk3::ApplicationWindow';

# Объявление переменных, которые будуд использоваться для
#  послеующего вызова функций:

my $window;
my $buffer;
my $file;

 sub new {
        $window = $_[0]; my $app = $_[1];
        $window = bless Gtk3::ApplicationWindow->new($app);
        $window->set_title ('FileChooserDialog Example');
        $window->set_default_size(400,400);
        $window->signal_connect( 'delete_event' => sub {$app->quit()} );

        # файл
        $file = undef;

        # виджет текстового поля вместе с буфером, в котором будет текст

        $buffer = Gtk3::TextBuffer->new();
        my $textview = Gtk3::TextView->new();
        $textview->set_buffer($buffer);
        $textview->set_wrap_mode('word');

        # виджет скроллинга для текстового поля

        my $scrolled_window = Gtk3::ScrolledWindow->new();
        $scrolled_window->set_policy('automatic', 'automatic');
        $scrolled_window->add($textview);
        $scrolled_window->set_border_width(5);

        # добавить скроллинг на базовое окно

        $window->add($scrolled_window);

        # the actions for the window menu, connected to the callback functions
        my $new_action = Glib::IO::SimpleAction->new('new',undef);
        $new_action->signal_connect('activate'=>\&new_callback);
        $window->add_action($new_action);

        my $open_action = Glib::IO::SimpleAction->new('open',undef);
        $open_action->signal_connect('activate'=>\&open_callback);
        $window->add_action($open_action);

        my $save_action = Glib::IO::SimpleAction->new('save',undef);
        $save_action->signal_connect('activate'=>\&save_callback);
```

```
	$window->add_action($save_action);

	my $save_as_action = Glib::IO::SimpleAction->new('save-as',undef);
	$save_as_action->signal_connect('activate'=>\&save_as_callback);
	$window->add_action($save_as_action);

	return $window;
}

# вызов для new

sub new_callback {
	my ($action, $parameter) = @_;
	$buffer->set_text('');
	print "New file created \n";
}

# вызов для open

sub open_callback {
	my ($action, $parameter) = @_;
	# создать видежет выбора файла с аргументами:
	#  : title of the window, parent_window, action
	# (buttons, response)
	my $open_dialog = Gtk3::FileChooserDialog->new('Pick a file',
	                                               $window,
	                                               'open',
	                                               ('gtk-cancel', 'cancel',
	                                               'gtk-open', 'accept'));

	# не только локальные файлы могут быть выбраны
	$open_dialog->set_local_only(FALSE);

	# показывать диалог выбора файла поверх всего приложения всегда
	$open_dialog->set_modal(TRUE);

	# сопоставить диалог открытия файла соответствующей
	# функции  open_response_cb()
	$open_dialog->signal_connect('response' => \&open_response_cb);


	# показать диалог
	$open_dialog->show();

}

# вызов функции диалога для open_dialog
sub open_response_cb {
	my ($dialog, $response_id) = @_;
	my $open_dialog = $dialog;
	# если ответ 'ACCEPTED' (кнопка 'Open' нажата)
	if ($response_id eq 'accept') {
```

```
                print "accept was clicked \n";

                # $file – файл, выбранный пользователем с помощью FileChooserDialog
                $file = $open_dialog->get_file();

                # загружаем содержимое файла в память:
                # успешно, если файл открылся и считался в память и флаги
                # для отлслеживания состояния файла, если он был изменен
                # в системе сторонними процессами

                my ($success, $content, $etags) = Glib::IO::File::load_contents($file) or print
"Error: Open failed \n";

                # ВНИМНИЕ: GIO читает и пишет файлы в raw формате, т.е.
                # sпобайтово, то есть это означает что запись проиводится без
                # перекодированияreads and writes files in raw bytes format,

                # Мы должны преобразовать эти данные так, чтобы Perl мог понять их..
                # Сперва мы должны преобразовать байты в поток  bytestring
                # без перекодировки

                $content = pack 'C*', @{$content};

                # Затем надо перекодировать bytestrng в utf8 кодировку

                my $content_utf8 = decode('utf-8', $content);

                # важно: необходимо знать длину файла в байтах, обычно
                # функция length имеет дело с логическими символами, но
                # не с физическими байтами. Для того, чтобы унать сколько
                # байтов преобразовано в кодировку UTF8, нужно использовать
                # функцию  length(Encode::encode_utf8($content) (для этого
                 # была испольована прагма 'use Encode'
                # [смотри больше о функции http://perldoc.perl.org/functions/length.html]
                # Альтернативные решения:
                # 1) Для вставки всего текста установите $buffer->set_text method
                #         to -1 (!текст должен быть ненулевой длины)
                # 2) В Perl Gtk3 указание длины текста крайне желательно!

                my $length = length(Encode::encode_utf8($content_utf8));
                $buffer->set_text($content_utf8, $length);

                my $filename = $open_dialog->get_filename();
                print "opened: $filename \n";

                $dialog->destroy();
                }
        # если ответ 'CANCEL' ( кнопка 'Cancel' нажата )
        elsif ($response_id eq 'cancel') {
                print "cancelled: Gtk3::FileChooserAction::OPEN \n";
                $dialog->destroy();
                }
```

```
}

# вызов функции для  save_as

sub save_as_callback {
	# создать filechooserdialog для сохранения:
	# the arguments are: title of the window, parent_window, action,
	# (buttons, response)
	my $save_dialog = Gtk3::FileChooserDialog->new('Pick a file',
                                                       $window,
                                                       'save',
                                                       ('gtk-cancel', 'cancel',
                                                       'gtk-save', 'accept'));
	# Диалог спросит подтверждения, если пользователь ввел в
	# качестве имени файла уже существующее имя файла

	$save_dialog->set_do_overwrite_confirmation(TRUE);

	# показывать диалог всегда поверх всего
	$save_dialog->set_modal(TRUE);

	if ($file) {
	  $save_dialog->select_file($file) or print "Error Selecting file failed\n";
	}

	# вызов функции для save_response_cb
	$save_dialog->signal_connect('response' => \&save_response_cb);

	# показать диалог
	$save_dialog->show();
}

# вызов функции для dialog save_dialog
sub save_response_cb {
	my ($dialog, $response_id) = @_;
	my $save_dialog = $dialog;

	# если  $response_id == 'ACCEPT' ( кнопка 'Save' нажата)
	if ($response_id eq 'accept') {
		# взять выделенный юзером файл
		# более соременно для этого использовать метод GFile $file->get_file (see above)
		$file = $save_dialog->get_file();

		# сохранить (смотри выше)
		save_to_file();

		# уничтожить FileChooserDialog
		$dialog->destroy;
	}
	elsif ($response_id eq 'cancel') {
		print "cancelled: FileChooserAction.SAVE \n";
		# уничтоить FileChooserDialog
```

```
                $dialog->destroy;
        }
}

# вызов функции для save
sub save_callback {
        my ($action, $parameter) = @_;
        # если $file существует
        if ($file) {
                save_to_file();
        }
        # $file – новый файл
        else {
                # испольуем save_as
                save_as_callback($action, $parameter);
        }
}

# вызов функции для  save_to_file
sub save_to_file {
        # прочитать содержимое буфера, без скрытых символов
        my ($start, $end) = $buffer->get_bounds();
        my $current_contents = $buffer->get_text($start, $end, FALSE);
        if ($current_contents) {


                # ВНИМАНИЕ СНОВА: Gio читает и пишет файлы в raw
                # bytes формате (см. выше)
                # поэтому метод replace_readwrite принимает ссылку на
                # массив , содежащий данные в raw bytes!!! Т.е. необходимо
                # преобразовать данные для перла в массиве:

                # во первых, необходимо преоразовать строку ввода в utf8 в
                # в bytestring
                my $content_utf8 = encode('utf-8', $current_contents);
                # Затем преобразовать bytestring в bytes и передать ссылку на массив функции
                my @contents = unpack 'C*', $content_utf8;

                # содержимое файла $file
                # аргументы: contents, etags, make_backup, flags, GError
                $file->replace_contents(\@contents,
                                        undef,
                                        FALSE,
                                        'none',
                                        undef) or print "Error: Saving failed\n";
                my $path = $file->get_path();
                print "saved $path \n";
        }
        # если содержимое пустое
        else {
                # создаем(если файл не существует) или перезаписываем в
                # режиме чтение/запись.
```

```
                # аргументы: etags, make_backup, flags, GError
                $file->replace_readwrite(undef,
                                         FALSE,
                                         'none',
                                         undef);
                my $path = $file->get_path();
                print "saved $path \n";
        }
}


# MAIN FUNCTION должен быть как можно меньше и не делать ничего, кроме
# создания вашего Gtk3::Application и его запуска
# Реальная работа должна быть всегда выполняться в ответ на сигналы, которые
# испускает Gtk3::Application.
# смотри ниже

use strict;
use warnings;
use utf8;

use Gtk3;
use Glib ('TRUE', 'FALSE');

my $app = Gtk3::Application->new('app.test', 'flags-none');

$app->signal_connect('startup'  => \&_init     );
$app->signal_connect('activate' => \&_build_ui );
$app->signal_connect('shutdown' => \&_cleanup  );

$app->run(\@ARGV);

exit;


# CALLBACK FUNCTIONS для SIGNALS испускаются главной(main) функцией.
# В этой функции делается “основная работа” (смотри ниже)

sub _init {
        my ($app) = @_;

        # действие "quit", вызывает соответствующую функцию
        my $quit_action = Glib::IO::SimpleAction->new('quit',undef);
        $quit_action->signal_connect('activate'=>\&quit_callback);
        $app->add_action($quit_action);

        # построить меню из конфигурационного файла XML
        # *.ui
        my $builder = Gtk3::Builder->new();
        $builder->add_from_file('12_3_filechooserdialog.ui') or die "file not found \n";
        my $menu = $builder->get_object('appmenu');
        $app->set_app_menu($menu);
```

```
}

sub _build_ui {
        my ($app) = @_;
        my $window = MyWindow->new($app);
        $window->show_all();
}

sub _cleanup {
        my ($app) = @_;
}

# обработать сигнал "quit"
sub quit_callback {
        print "You have quit \n";
        $app->quit();
}
```

Используемые методы для виджета FileChooserDialog

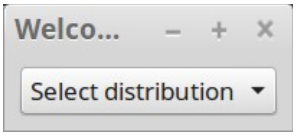
Отметьте что действие *action* FileChooserDialog может быть одним из перечисленных: *"open"* (диалог позволяет выбрать пользователю только существующий файл), *"save"* (диалог позволяет выбрать пользователю только существующий файл или ввести новое имя файла), *"select_folder"* (позволяет выбрать директорию), *"create_folder"* (позволяет выбрать или создать новую директорию).

Помимо методов и функций используемых в коде, Gtk3-Perl позволяет сделать еще некоторые действия:

- *set_show_hidden(TRUE)* - показать скрытые файлы и директории.
- *set_select_multiple(TRUE)* позволяет выбрать несколько файлов одновременно. Это актуально в том случае, если действие или возможности интерфейса определены как *"open"* или *"select_folder"*.
- В диалоге 'Save as', *set_current_name(current_name)* присваивает переменной *current_name* имя файла, введеного пользователем; current_name модет быть например Untitled.txt. Этот метод не используется в диалоге "Save as".
- Допустим текущая директория "recent items". Для установки другой директории можно испольовать метод *set_current_folder_uri(uri)*; но обратите внимание на то, что вы должны использовать этот метод показывать определенную папку только тогда, когда используется команда "Save as" и файл уже где нибудь сохранен.

13.Дерево каталогов и меню выбора (TreeViews and ComboBoxes (using the M/V/C design))

13. 1. Виджет ComboBox (one column)



ComboBox выводит в терминал выбранную пользователем позицию выкидного меню.

Код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my @distros = ('Select distribution', 'Fedora', 'Mint', 'Suse');

my $window=Gtk3::Window->new('toplevel');
$window->set_title('Welcome to GNOME');
$window->set_default_size(200,-1);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

# данные имеют тип строк
my $liststore = Gtk3::ListStore->new('Glib::String');

# добавляем данные в модель
foreach my $data (@distros) {

        # Можно добавить данные двумя различными способами
        # 1) обавить пустую строку в Treestore; и для неё
        # создать указатель или ссылку ($iter)
        # 2) Для заполнения строки можно использовать Gtk3::Treestore
        #
        # example: $liststore ->set ($iter, 0 => 'content of the row in column 1',
        # 1 => ('content of the row in column 2 etc.)

        my $iter = $liststore->append();
        $liststore->set($iter, 0 => "$data");

        }

# добавляем комбо бокс
my $combobox = Gtk3::ComboBox->new_with_model($liststore);
```

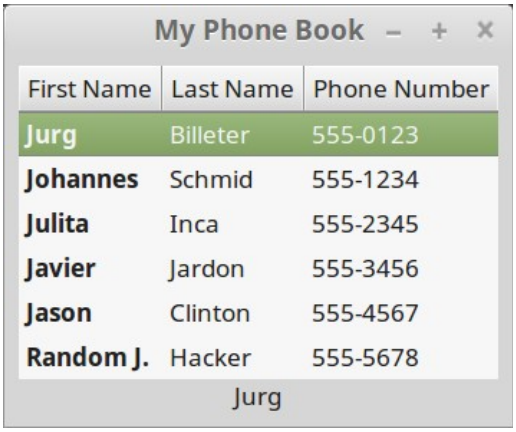
```
# показываем текст
my $cell = Gtk3::CellRendererText->new();

# устанавливаем ячейку комбобокса в начало

$combobox->pack_start($cell, FALSE);
# связываем свойство ('text') с ячейкой (cell)
# в колонку (column 0) в модели, используемой combobox
$combobox->add_attribute($cell, 'text', 0);

# первая строка - активная по умолчанию вначале
$combobox->set_active(0);

# вызов функции в зависимости от выбранной поиции
$combobox->signal_connect('changed'=>\&on_changed);

# добавляем выкидное меню на окно
$window->add($combobox);

# показываем всё
$window->show_all;
Gtk3->main();

sub on_changed {
        # если выбранная строка не является первой, пишем в терминал
        my ($combo) = @_;
        my $active = $combo->get_active();
        if ($active != 0) {
                print "You choose $distros[$active]. \n";
                }
        return TRUE;
        }
```

Используемые методы для виджета ComboBox

Виджет ComboBox работает в представлении модели Model/View/Controller: Model хранит данные; View получает уведомления об изменении и выводит на экран содержание модели; the Controller, наконец, изменяет состояние модели и уведомляет об этих изменениях. Для более детальной информации об этом предсавлении и используемых методах ComboBox, смотри The Model/View/Controller .

В строке 54 сигнал *'changed'* выывает функцию *on_changed()* используя *$widget->signal_connect("signal" => \&callback function)*. См. сигналы и функции для более детального понимания.

13. 2. TreeView with ListStore



Виджет TreeView покаывает простой ListStore с выделением определенной позиции в списке которое вызывает сигнал 'changed'.

Код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my @columns = ('First Name','Last Name','Phone Number');
my @phonebook =(['Jurg', 'Billeter', '555-0123'],
                ['Johannes', 'Schmid', '555-1234'],
                ['Julita', 'Inca', '555-2345'],
                ['Javier', 'Jardon', '555-3456'],
                ['Jason', 'Clinton', '555-4567'],
                ['Random J.', 'Hacker', '555-5678']);

my $window=Gtk3::Window->new('toplevel');
$window->set_title('My Phone Book');
$window->set_default_size(200,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

# данные
# (три текста для каждой строки по три для каждой колонки)
my $listmodel = Gtk3::ListStore->new('Glib::String','Glib::String','Glib::String');

# append the values in the model
for (my $i=0; $i <= $#phonebook; $i++) {
        # Можно добавить данные двумя различными способами
        # 1) обавить пустую строку в Treestore; и для неё
        # создать указатель или ссылку ($iter)
        # 2) Для заполнения строки можно использовать Gtk3::Treestore

        # example: $liststore ->set ($iter, 0 => 'content of the row in column 1',
```

```
	# 1 => ('content of the row in column 2 etc.)

	my $iter = $listmodel->append();
	$listmodel->set($iter, 0 => "$phonebook[$i][0]",
				1 => "$phonebook[$i][1]",
				2 => "$phonebook[$i][2]");
	}

# заполняем treeview
my $view = Gtk3::TreeView->new($listmodel);

# показываем текст для каждой из трех колонок
for (my $i=0; $i <= $#columns; $i++) {
	my $cell = Gtk3::CellRendererText->new();

	# выставляем фонт для первой колонки
	if ($i == 0) {
		$cell->set_property('weight_set',TRUE);
		# !!! Pango.Weight.BOLD is not recognized in perl I don't know
		# why. I took instead 800 for bold (default value (ie normal) is 400)
		$cell->set_property('weight',800);
		}

	# колонка содана
	# использование:
	# Gtk3::TreeViewColumn->new_with_attributes (title, cell_renderer,
	# attr1 => col1, ...)
	my $col = Gtk3::TreeViewColumn->new_with_attributes($columns[$i],$cell,'text' => $i);

	# иначе можно сделать подобно примеру выше:
	# первое: создаем виджет TreeViewColumn
	#my $col = Gtk3::TreeViewColumn->new();
	# второе: показваем текст в начале колонки,
	#
	#$col->pack_start($cell, FALSE);
	# третье: устанавливаем заголовок
	#$col -> set_title($columns[$i]);
	# четвертое: устанавливаем аттрибуты
	#	USAGE: add_attribute(cell_renderer, attribute=>column);
	#$col->add_attribute($cell, text=>$i);

	# добавляем в treeview
	$view->append_column($col)
	}

# создаем объект TreeSelection
my $treeselection=$view->get_selection();
# когда строка выделена, испускаем сигнал
$treeselection->signal_connect('changed' => \&on_changed);

# показываем на лейбле выделение
my $label = Gtk3::Label->new();
```

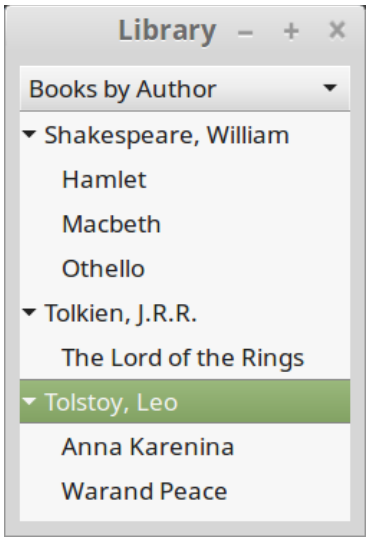
```
$label->set_text('');

# добавляем видеты на сетку
my $grid = Gtk3::Grid->new();
$grid->attach($view, 0,0,1,1);
$grid->attach($label, 0,1,1,1);

# сетку на базовое окно
$window->add($grid);

# показать все
$window->show_all;
Gtk3->main();

sub on_changed {
	my ($sel) = @_;

	# получить модель и укаатель на данные в модели
	my ($model, $iter) = $sel->get_selected();

	# установить аголовок в ависимости от выделения
	if ($iter != '') {
		my $value = $model->get_value($iter,0);
		$label->set_text("$value");
		}
	else {
		$label->set_text('');
		}
	return TRUE;
	}
```

Используемые методы для виджета TreeView

Виджет TreeView работает в представлении модели Model/View/Controller: Model хранит данные; View получает уведомления об изменении и выводит на экран содержание модели; the Controller, наконец, изменяет состояние модели и уведомляет об этих изменениях. Для более детальной информации об этом предсавлении и используемых методах TreeModel, смотри The Model/View/Controller .

В строке 83 сигнал *'changed'* вызывает функцию *on_changed()* используя *$widget->signal_connect("signal" => \&callback function)*. См. сигналы и функции для более детального понимания.

13. 3. Простой показ дерева каталогов с виджетом TreeStore



Виджет TreeView , показывающий TreeStore.

Код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my @books = (['Tolstoy, Leo', 'Warand Peace', 'Anna Karenina'],
                    ['Shakespeare, William', 'Hamlet', 'Macbeth', 'Othello'],
                    ['Tolkien, J.R.R.', 'The Lord of the Rings']);

my $window=Gtk3::Window->new('toplevel');
$window->set_title('Library');
$window->set_default_size(250,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

# данные в модели M/V/C
# создать treestore с одной колонкой
my $store = Gtk3::TreeStore->new('Glib::String');

# присвоить переменные  model
for (my $i=0; $i <= $#books; $i++) {
        # Можно добавить данные двумя различными способами
        # 1) обавить пустую строку в Treestore; и для неё
        # создать указатель или ссылку ($iter)
        # 2) Для заполнения строки можно использовать Gtk3::Treestore

        # example: $liststore ->set ($iter, 0 => 'content of the row in column 1',
        # 1 => ('content of the row in column 2 etc.)

        my $iter = $store->append();
        $store->set($iter, 0 => "$books[$i][0]");
```

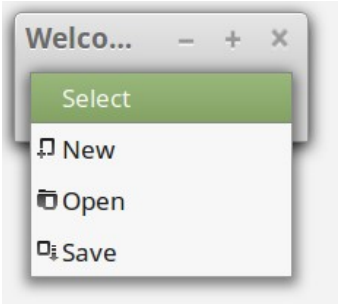
```
        for (my $j=1; $j <= $#{$books[$i]}; $j++) {
                # in dieser Spalter fügen wir Kind Iters zu den Eltern Iters hinzu
                # und fügen diesen Kind Iters Daten hinzu / erneut nur 1 Spalte
                my $iter_child = $store->append($iter);
                $store->set($iter_child, 0 => "$books[$i][$j]");
                }
        }

my $view = Gtk3::TreeView->new();
$view->set_model($store);

my $renderer_books = Gtk3::CellRendererText->new();
my $column_books = Gtk3::TreeViewColumn->new_with_attributes('Books by Author',
$renderer_books, 'text'=>0);
$view->append_column($column_books);
$column_books->set_sort_column_id(0);

$window->add($view);
$window->show_all;
Gtk3->main();
```

Используемые методы для виджета TreeView

Виджет TreeView работает в представлении модели Model/View/Controller: Model хранит данные; View получает уведомления об изменении и выводит на экран содержание модели; the Controller, наконец, изменяет состояние модели и уведомляет об этих изменениях. Для более детальной информации об этом предсавлении и используемых методах TreeView, смотри The Model/View/Controller .

13. 4. The Model/View/Controller design (to do)

13. 5. виджет ComboBox (two columns)



Выводит на печать выделенное меню.

код

```
#!/usr/bin/perl
```

```
use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my @actions = (        ['Select', ''],
                       ['New', 'document-new'], # same as 'gtk-new'
                       ['Open', 'document-open'], # same as 'gtk-open'
                       ['Save', 'document-save']); # same as 'gtk-save'

my $window=Gtk3::Window->new('toplevel');
$window->set_title('Welcome to GNOME');
$window->set_default_size(200,-1);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

# the data in the model, of type string on two columns
my $listmodel = Gtk3::ListStore->new('Glib::String','Glib::String');

# append the data
for (my $i; $i<=$#actions; $i++) {

        # Можно добавить данные двумя различными способами
        # 1) обавить пустую строку в Treestore; и для неё
        # создать указатель или ссылку ($iter)
        # 2) Для заполнения строки можно использовать Gtk3::Treestore
        # example: $liststore ->set ($iter, 0 => 'content of the row in column 1',
        # 1 => ('content of the row in column 2 etc.)

        my $iter = $listmodel->append();
        $listmodel->set($iter, 0 => "$actions[$i][0]", 1 => "$actions[$i][1]");

        }

my $combobox = Gtk3::ComboBox->new_with_model($listmodel);
my $renderer_pixbuf = Gtk3::CellRendererPixbuf->new();
my $renderer_text = Gtk3::CellRendererText->new();

$combobox->pack_start($renderer_pixbuf, FALSE);
$combobox->pack_start($renderer_text, FALSE);

$combobox->add_attribute($renderer_text, 'text' => 0);
$combobox->add_attribute($renderer_pixbuf, 'icon-name' => 1);

$combobox->set_active(0);

$combobox->signal_connect('changed', \&on_changed);

$window->add($combobox);
$window->show_all;
Gtk3->main();
```

```
sub on_changed {
        my ($combo) = @_;
        my $active = $combo->get_active();
        if ($active != 0) {
                print "You choose $actions[$active][0]. \n";
                }
        return TRUE;
        }
```

Используемые методы для виджета ComboBox

Виджет TreeView работает в представлении модели Model/View/Controller: Model хранит данные; View получает уведомления об изменении и выводит на экран содержание модели; the Controller, наконец, изменяет состояние модели и уведомляет об этих изменениях. Для более детальной информации об этом предсавлении и используемых методах TreeView, смотри The Model/View/Controller .

Сигнал 'changed' выывает функцию on_changed() используя widget.connect(signal, callback function). См. сигналы и функции для более детального понимания.

13. 6. Комбинации виджетов Treeview м ListStore

Этот TreeView показывает ListStore при выделении в котором испускается сигнал 'changed' .

код

```
#!/usr/bin/perl

use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my @list_of_dvds = ('The Usual Suspects','Gilda','The Godfather', 'Pulp Fiction', 'Once Upon a
Time in the West', 'Rear Window');

my $window=Gtk3::Window->new('toplevel');
$window->set_title('My DVDs');
```

```
$window->set_default_size(250,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

my $listmodel = Gtk3::ListStore->new('Glib::String');

my $row_count = 0;

for (my $i=0; $i <= $#list_of_dvds; $i++) {
        my $iter = $listmodel->append();
        $listmodel->set($iter, 0 => "$list_of_dvds[$i]");
        # Wenn eine Zeile hinzugefügt wurde, erhöhe die Variable $row_count
        $row_count++;
        }

my $view = Gtk3::TreeView->new($listmodel);
my $cell = Gtk3::CellRendererText->new();
my $col = Gtk3::TreeViewColumn->new_with_attributes('Title',$cell,'text' => 0);
$view->append_column($col);

my $selection=$view->get_selection();
$selection->signal_connect('changed' => \&on_changed);

my $label = Gtk3::Label->new();
$label->set_text('');

my $button_add = Gtk3::Button->new('Add');
$button_add->signal_connect('clicked' => \&add_cb);

my $entry = Gtk3::Entry->new();
my $button_remove = Gtk3::Button->new('Remove');
$button_remove->signal_connect('clicked'=>\&remove_cb);

my $button_remove_all = Gtk3::Button->new('Remove All');
$button_remove_all->signal_connect('clicked' => \&remove_all_cb);

my $grid = Gtk3::Grid->new();
$grid->attach($view, 0, 0, 4, 1);
$grid->attach($label, 0, 1, 4, 1);
$grid->attach($button_add, 0, 2, 1, 1);
$grid->attach_next_to($entry, $button_add, 'right', 1, 1);
$grid->attach_next_to($button_remove, $entry, 'right', 1, 1);
$grid->attach_next_to($button_remove_all, $button_remove, 'right', 1, 1);

$window->add($grid);
$window->show_all;
Gtk3->main();

sub on_changed {
        my ($sel) = @_;
        my ($model, $iter) = $sel->get_selected();
```

```
	if ($iter != '') {
		my $value = $model->get_value($iter,0);
		$label->set_text("$value");
		}
	else {
		$label->set_text('');
		}
	return TRUE;
	}

sub add_cb {
	my $title = $entry->get_text();
	my $add_iter = $listmodel->append();
	$listmodel->set($add_iter, 0 => "$title");
	# and print a message in the terminal
	print "$title has been added \n";
	# Wenn eine Zeile hinzugefügt wurde, erhöhe die Variable $row_count
	$row_count++;
	}

sub remove_cb {
	# check if there is still an entry in the model
	# the methode iter_n_children($iter) returns the number of children
	# that iter has or here with $iter=NULL the number of toplevel nodes
	# which are in a liststore (no childs!!) the number of lines
	my $len = $listmodel->iter_n_children();
	#if ($len != 0) {
	# another way is the methode $listmodel->get_iter_first which returns
	# FALSE if the tree is empty
	#if ($listmodel->get_iter_first()) {

	# last but not least you can save the lenght of rows in an own
	# variable (here: $rows (see below)
	if ($row_count != 0) {
		# get the selection
		my ($model, $iter) = $selection->get_selected();
		# if there is a selection, print a message in the terminal
		# and remove it from the model (TO DO)

		# we want the data at the model's column 0
		# where the iter is pointing
		my $value = $model->get_value($iter,0);
		print "$value has been removed \n";
		$listmodel->remove($iter);

		# wenn eine Zeile gelöscht wurde, erniedrige die Variable $row_count
		$row_count--;
		}
	else {
		print "Empty list \n";
		}
	}
```

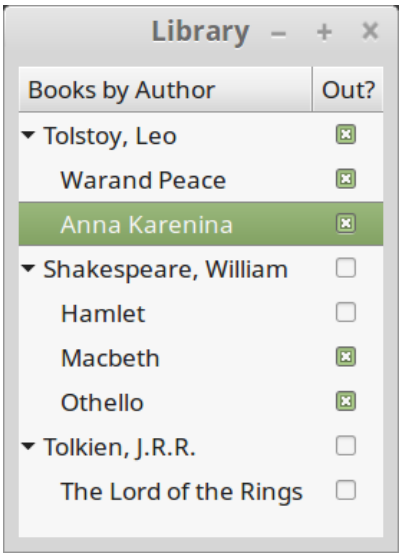
```
sub remove_all_cb {
	# check if there is still an entry in the model
	if ($row_count != 0) {
		# remove all the entries in the model
		for (my $i=1; $i <= $row_count; $i++) {
			my $iter = $listmodel->get_iter_first;
			$listmodel->remove($iter);
			}

		# Setze die Anzahl der Zeilen auf 0
		$row_count = 0;
		}
	else {
		print "Empty list \n";
		}
	}
```

Useful methods for a TreeView widget

Виджет TreeView работает в представлении модели Model/View/Controller: Model хранит данные; View получает уведомления об изменении и выводит на экран содержание модели; the Controller, наконец, изменяет состояние модели и уведомляет об этих изменениях. Для более детальной информации об этом предсавлении и используемых методах TreeView, смотри The Model/View/Controller

Сигнал 'changed' выывает функцию on_changed() используя widget.connect(signal, callback function). См. сигналы и функции для более детального понимания.

13. 7. More Complex TreeView with TreeStore



This TreeView displays a TreeStore with two columns, one of which is rendered as a toggle.

Code used to generate this example

```
#!/usr/bin/perl
```

```
use strict;
use Glib ('TRUE','FALSE');
use Gtk3 -init;

my @books =(['Tolstoy, Leo', ['Warand Peace', TRUE], ['Anna Karenina', FALSE]],
                ['Shakespeare, William', ['Hamlet', FALSE], ['Macbeth', TRUE], ['Othello', FALSE]],
                ['Tolkien, J.R.R.', ['The Lord of the Rings', FALSE]],);

my $window=Gtk3::Window->new('toplevel');
$window->set_title('Library');
$window->set_default_size(250,100);
$window->set_border_width(10);
$window->signal_connect('delete_event' => sub {Gtk3->main_quit});

# the data are stored in the moel
# create a treestore with two column
my $store = Gtk3::TreeStore->new('Glib::String', 'Glib::Boolean');

# fill in the model
for (my $i=0; $i <= $#books; $i++) {
        # You add data in two steps
        # 1) Add a new empty row to the Treestore; to this row
        # generate a reference or a pointer ($iter)
        # 2) In order to fill this row with content, you apply the Gtk3::Treestore
        # set-methode to this
        # !!! The list of pairs must contain as many elements as
        #  the number of columns in the  Tree- or ListStore!!!
        # example: $liststore ->set ($iter, 0 => 'content of the row in column 1',
        # 1 => ('content of the row in column 2 etc.)

        # First of all you generate the parent Iter and fill in the parent cells.
        # these are always at in the first place (i.e. $books[$i][0]).
        # only one column is needed
        my $iter = $store->append();
        $store->set($iter, 0 => "$books[$i][0]", 1 => FALSE);

        for (my $j=1; $j <= $#{$books[$i]}; $j++) {
                # in dieser Spalter fügen wir Kind Iters zu den Eltern Iters hinzu
                # und fügen diesen Kind Iters Daten hinzu / erneut nur 1 Spalte
                my $iter_child = $store->append($iter);
                $store->set($iter_child, 0 => "$books[$i][$j][0]", 1 =>"$books[$i][$j][1]");
                }
        }

# the treeview shows the model
# create a treeview on the model $store
my $view = Gtk3::TreeView->new();
$view->set_model($store);

# the cellrenderer for the first column - text
my $renderer_books = Gtk3::CellRendererText->new();
# the first column is created
```

```
my $column_books = Gtk3::TreeViewColumn->new_with_attributes('Books by Author',
$renderer_books, 'text'=>0);
# and it is appended to the treeview
$view->append_column($column_books);

# the books are sortable by authors
$column_books->set_sort_column_id(0);

# the cellrenderer for the second column - boolean renderer as a toggle
my $renderer_in_out = Gtk3::CellRendererToggle->new();
# the second column is created
my $column_in_out = Gtk3::TreeViewColumn->new_with_attributes('Out?', $renderer_in_out,
'active'=>1);
# and it is appended to the treeview
$view->append_column($column_in_out);
# connect the cellrenderertoggle with a callback function
$renderer_in_out->signal_connect('toggled' => \&on_toggled);

# add the treeview to the window
$window->add($view);

# show the window and run the Application
$window->show_all;
Gtk3->main();

# callback function for the signa emitted by the cellrenderertoggle
sub on_toggled {
        my ($widget, $path_string) = @_;

        # Get the boolean value (1=TRUE, 0=FALSE) of the selected row
        # first generate a Gtk3::TreePath, by using $path_string as a argument
        # to the new_from_string method of Gtk3::TreePath.
        # This will give us a 'geographical' indication which row was edited.
        my $path = Gtk3::TreePath->new_from_string($path_string);
        # the get the Gt3::Treeiter of the TreePath $path, which will refer
        # to a row
        my $iter = $store->get_iter($path);
        # last get the value with the function get_value on the model
        my $current_value = $store->get_value($iter,1);

        # change the value of the toggled item
        # instead of the if/elsif construction you can simple write this line
        # "$current_value ^= 1;" [but I don't understand why this works :-)]
        if ($current_value == 0) {$current_value = 1;}
        elsif ($current_value==1){$current_value = 0; }
        $store->set( $iter, 1, $current_value);

        # check if length if the path is 1
        # (that is, if we are selecting an author (= parent cell)
        if (length($path_string) == 1) {

                # get the number of the childrens that the parent $iter has
```

```
		my $n = $store->iter_n_children($iter);

		# get the iter associated with its first child
		my $citer = $store->iter_children($iter);

		foreach (my $i = 0; $i <= $n-1; $i++) {
			$store->set($citer,1 => $current_value);
			$store->iter_next($citer);
			}
		}

	# if the length of the path is not 1
	# (that is if we are selecting a book)
	else {
		# get the parent and the first child of the parent
		# (that is the first book of the author)
		my $piter = $store->iter_parent($iter);
		my $citer = $store->iter_children($piter);

		# get the number of the childrens that the parent $iter has
		my $n = $store->iter_n_children($piter);

		# Erzeuge eine Variable, mit der mittels einer Schleife
		#überprüft wird, ob alle Kinder items selected sind
		my $all_selected;

		# check if all children are selected
		foreach (my $i = 0; $i <= $n-1; $i++) {
			my $value = $store->get_value($citer,1);
			if ($value == 1) {
				$all_selected = 1;
				}
			if ($value == 0) {
				$all_selected = 0;
				last;
				}
			$store->iter_next($citer);
			}

		# wenn all_selected = 1 (=TRUE) soll auch das Eltern item
		# ausgewählt werden
		if ($all_selected == 1) {
			$store->set($piter, 1, 1);
			}
		# wenn ich alle Kind Elemente selektiert sind
		# soll auch das Eltern Element nicht selektiert sein
		elsif ($all_selected == 0) {
			$store->set($piter, 1, 0);
			}
		}
```

```
        }
```

Useful methods for a TreeView widget

Виджет TreeView работает в представлении модели Model/View/Controller: Model хранит данные; View получает уведомления об изменении и выводит на экран содержание модели; the Controller, наконец, изменяет состояние модели и уведомляет об этих изменениях. Для более детальной информации об этом предсавлении и используемых методах TreeView, смотри The Model/View/Controller.

Сигнал 'toggle' выывает функцию on_toggled() используя widget.connect(signal, callback function). См. сигналы и функции для более детального понимания.

Вопросы?

Если вы нашли программные ошибки, опечатки или другие ошибки – дайте мне знать (Maximilian-Lika@gmx.de)

Если Вы имеете каке либо вопросы относительно ипользования задавайте их в лист рассылки (gtk-perl-list@gnome.org) здесь наиболее вероятно получить ответ.