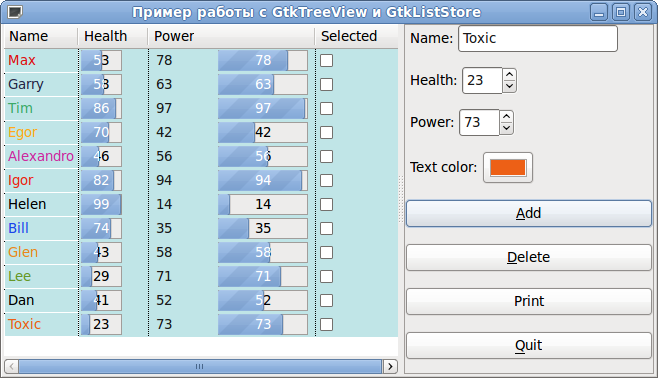
Работа с GtkTreeView и GtkListStore с помощью редактора Glade для начинающих

[Программирование\*](https://habr.com/ru/hub/programming/)

В этом посте я хочу рассказать про работу с очень интересными виджетами (объектами) библиотеки графических интерфейсов GTK+: **GtkTreeView** и **GtkListStore**. GtkTreeView — виджет для отображения деревьев и списков. GtkListStore — виджет представляющий модель списка.  
Создавать их я буду с помощью редактора Glade, в интернете очень мало материала именно по работе с виджетами GTK+ (и особенно с этими) с помощью этого редактора интерфейсов. Я уже писал немного про работу с ним. Те, кто никогда с ним не работал — советую прочитать [этот](http://habrahabr.ru/blogs/development/107403/) пост.  
  
Возможностей работы с этими виджетами очень много и я планирую написать еще несколько статей по этой теме (если это конечно заинтересует читателей). Этот пост расчитан на начинающих еще только знакомиться с библиотекой GTK+ и её возможностями. Поэтому сегодня я рассмотрю достаточно простой пример.  
  
Смысл его будет в том, что будем добавлять данные в объект класса GtkListStore (я буду иногда называть его хранилищем) о персонаже компьютерной игры и выводить их на экран в наглядной форме с помощью GtkTreeView.

  
  
Но добавлять в GtkListStore будем не только информацию о персонаже. Это хранилище позволяет хранить различные данные. Например стили шрифта, имя шрифта, цвета, картинки, различные настройки виджетов. В данном примере я покажу как можно для каждого отдельного персонажа задать отдельно цвет его написания в GtkTreeView (его еще называют виджет просмотра дерева, хотя в данном примеры мы будем просмотривать структуру списочного формата, а не древовидного).  
  
Начнём с того, что рассмотрим хранилище данных — виджет GtkListStore. Он представляет структуру данных похожую на таблицу (точно так же как таблица в базе данных) каждая строка это запись, запись состоит из данных (атомарных) различных типов. Каждая колонка задаёт свой тип данных. Колонки идентифицируются порядковым номером: первая колонка — 0, вторая колонка — 1, третья колонка — 2 и т.д.  
  
Обычно создают перечисление для более понятного обращения к колонкам хранилища:

**enum**

{

NAME = 0,

HEALTH,

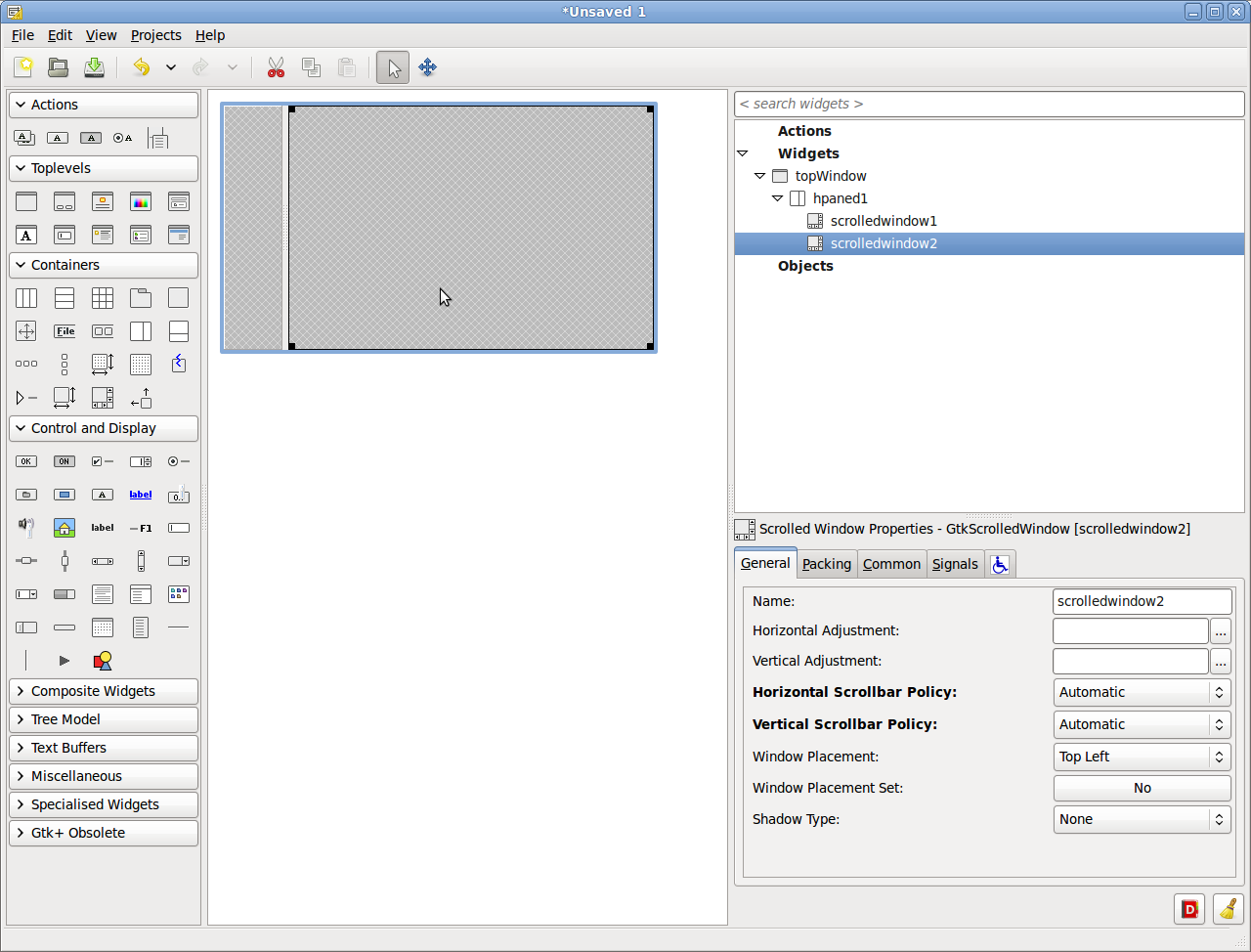
POWER,

SELECTED,

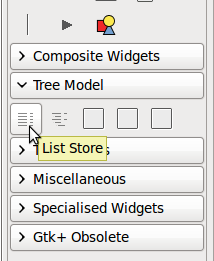
FONT\_COLOR

};

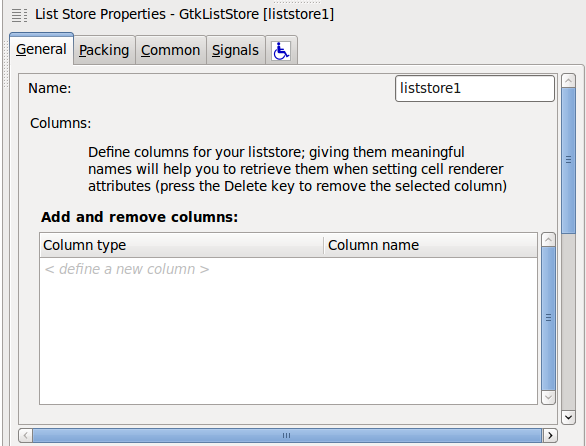
Для начала открываем Glade (я для примера буду использовать версию 3.6.7 — из репозитория Ubuntu, хотя есть более новые версии, поддерживающие GTK+ 3.0). Создаём форму, слева в панеле объектов ищем Window и кладём на рабочую область. Слева на панелей свойств во вкладке General ищем Window Title и задаём его как: *«Пример работы с GtkTreeView и GtkListStore»*, во вкладке Common задаём Width request и Height requst: *650* и *350* соответственно. Далее нужно будет положить на неё Horizontal Panes, в два отсека которых кладутся Scrolled Window. У Вас должно получится на данном шаге, как показано картинке ниже (кликабельно):



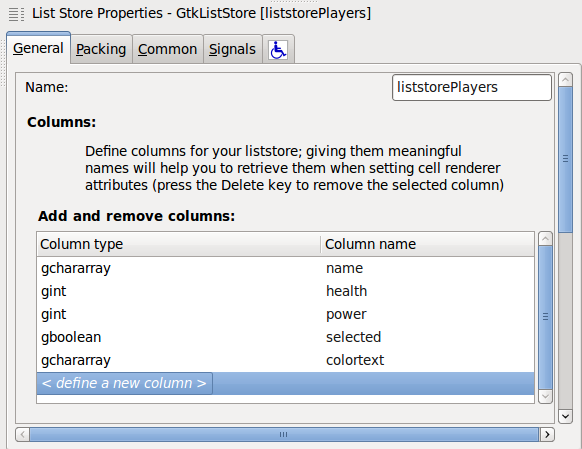
теперь идём в панель объектов нажимаем на Tree Model и ищем там List Store — наше будущее хранилище (виджет GtkLitStore).



По нажатию по этому виджету он автоматически появится в дереве виджетов справо во вкладке Objects. Нажмаем на него и смотрим на свойства. Отобразятся вот такие настройки, правда можно еще ниже прокрутить и там будет Add and remove rows, но нам это не нужно, т.к. данные будем вводить из контролов (которые сделаем чуть позже).



Во-первых, меняем имя на *liststorePlayers*. Во-вторых, в Add and Remove Columns нам нужно добавить следующие колонки: Имя (name) — строка, Здоровье (неalth) — целочисленный, Сила (power) — целочисленный, Флаг выделения (seleted) — логический и цвет шрифта (colortext) — тип цвета.  
  
Итак добавляем первую колонку имя — выбираем тип *gchararray* в Column type, а в Column name пишем *name*. Далее вторая колонка здоровье — в Column type пишем *gint*, в Column name *health*. Аналогично для *power* типа *gint*. Далее флаг выделение — выбираем тип *gboolean*, а в Column name *selected*. А далее… А далее нас ждёт небольшая неприятность. Нам нужен для цвета шрифта типа GdkColor (структура, используемая в GTK+ для представления цвета). Вы посмотрите на то большое количество возможных вариантов выбора типа, но GdkColor там нет. Видимо это недоработка разработчиков Glade и прийдёт добавить нам самим. Если попробуем вписать его туда — всё равно ничего не получится. Нужно вспомнить, что Glade сохраняет описание интефейса в XML файле, а следовательно его можно будет исправить. Значит, создаём еще одно поле типа *gchararray*, например, в Column type пишем *colortext*.



Сохраняем наше описание интерфейса с именем mainForm и открываем полученный файл. Ищем там GtkListStore.

<**object** class="GtkListStore" id="liststorePlayers">

<**columns**>

<!-- column-name name -->

<**column** type="gchararray"/>

<!-- column-name health -->

<**column** type="gint"/>

<!-- column-name power -->

<**column** type="gint"/>

<!-- column-name selected -->

<**column** type="gboolean"/>

<!-- column-name colortext -->

<**column** type="gchararray"/>

</**columns**>

</**object**>

Меняем наш *gchararray* на *GdkColor*

<**object** class="GtkListStore" id="liststorePlayers">

<**columns**>

<!-- column-name name -->

<**column** type="gchararray"/>

<!-- column-name health -->

<**column** type="gint"/>

<!-- column-name power -->

<**column** type="gint"/>

<!-- column-name selected -->

<**column** type="gboolean"/>

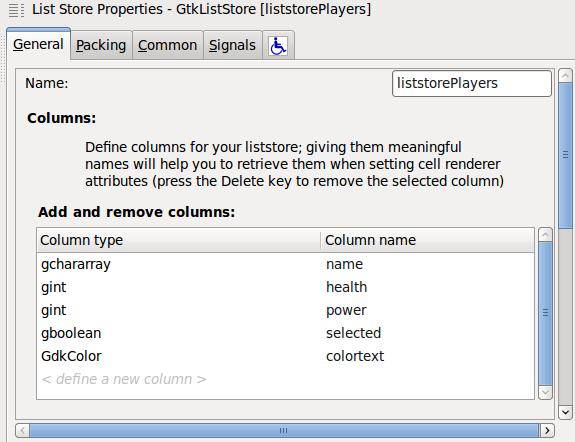
<!-- column-name colortext -->

<**column** type="GdkColor"/>

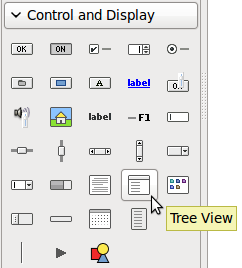
</**columns**>

</**object**>

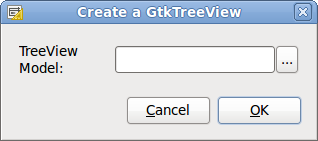
Сохраняемся и открываем снова в Glade. Смотрим тип колонки и видим, что всё так, как нам и нужно:



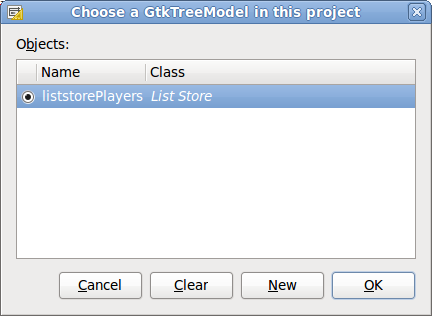
Настройка хранилища GtkListStore на этом завершена. Теперь нам необходимо сделать так, чтобы данные можно было отображать в более менее красивой форме. Для этого идём снова в панель объектов и ищем там Tree View — это виджет GtkTreeView.



Кладём его на левый scrolledwindow (у меня он называется scrolledwindow1, Вы можете вообще все объекты переименовывать, но я переименовываю только те, которые будут потом в коде использоваться, вообще желательного все переименовывать). После того как Вы взяли Tree View на панели объектов и щелкнув по левому scrolledwindow Вам вылетит сообщение о том, что следует указать TreeView Model.



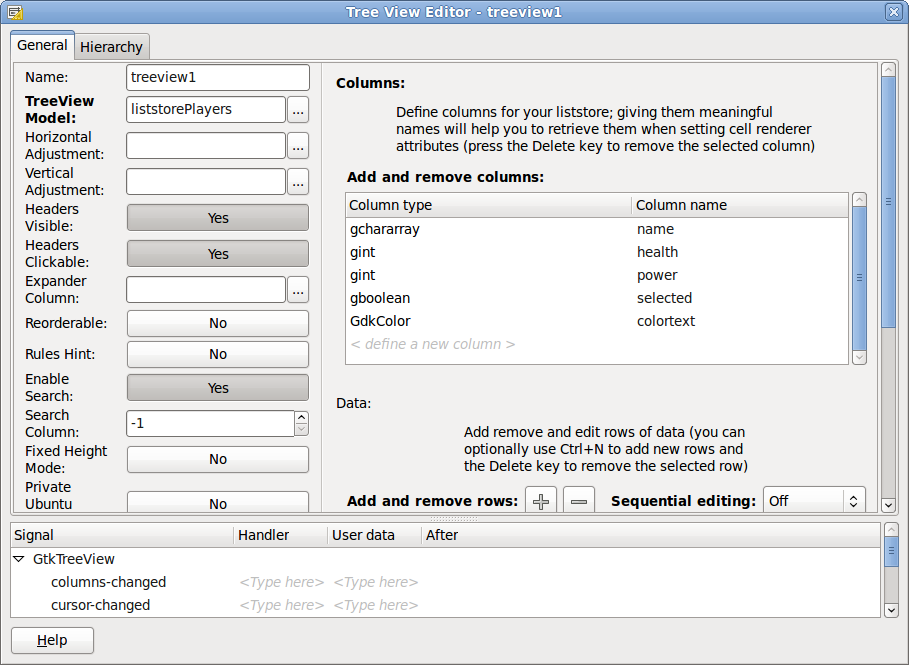
В нашем случае TreeView Model это и есть наше хранилище — GtkListStore, а именно виджет liststorePlayers. Нажимаем **...** и выбираем там *liststorePlayers*.



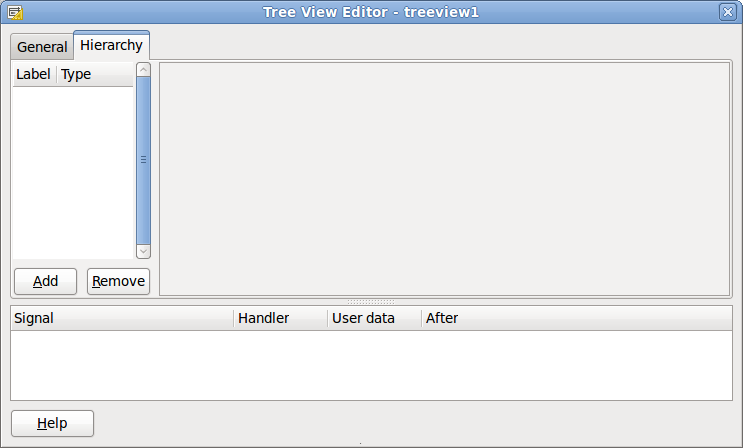
Нажимаем OK для этого диалога и для предыдущего. Теперь нам необходимо настроить или правильнее будет сказать связать данные из хранилища с конкретного формой отображения их на экране. Нажимаем на treeview1 (а именно так он у Вас называется) на правой панеле объектов и идём в верхнее меню, где ищем Edit.



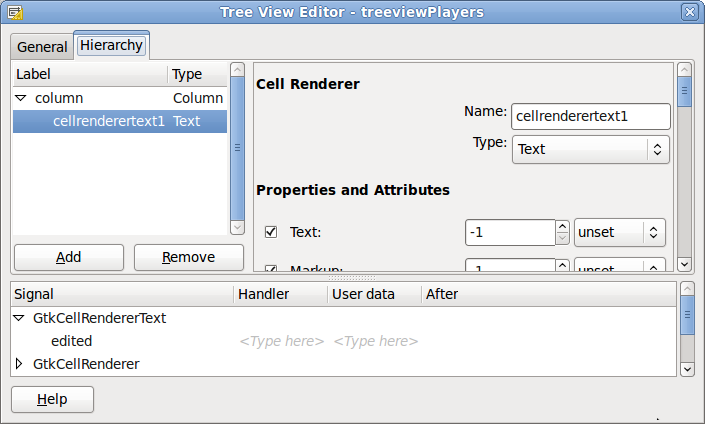
Откроется диалог редактирования нашего Tree View.



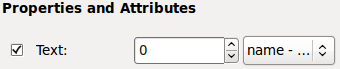
Во вкладке General находятся общие свойства для всего Tree View в целом. Во-первых, меняем Name на *treeviewPlayers*. А так же свойство Enable Grid Lines ставим в *Horizontal and Vertical* — отображение визуального разделения строк и колонок. Переходим во вкладку Hierarcy. В ней как раз мы и будем связать данные из хранилища с конкретной реализацией их отображения.



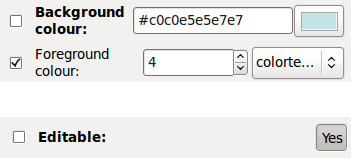
Немножко теории. Запомните, что колонки в List Store и колонки в Tree View — это абсолютно разные вещи и никакой связи между ними нет. Конкретные данные привязывают не к колонкам Tree View. Колонки Tree View больше служат каким-то логическо-визуальным разделением для выводимых данных. Для представления конкретных данных из хранилища используются объекты класса **GtkCellRenderer**.  
  
В нашем примере будет использоваться 4 различных типов объектов представления:  
**GtkCellRendererText** — обычный текст, для отображения name из хранилища;  
**GtkCellRendererProgress** — шкала выполнения, а если более просто — это обычный ProgressBar, с помощью него мы будем отображать health и power из хранилища;  
**GtkCellRendererSpin** — кнопка «карусель», с помощью неё мы сможем сможем изменять значение power прямо в Tree View;  
**GtkCellRendererToggle** — кнопку переключения в ячейке, c помощью неё будем отображать флаг выделения записи.  
  
Возвращаемся в Glade. Нажимаем кнопку Add и у нас появится одна колонка. Вообще можно колонок будет не добавлять в принципе. Одной хватит, но будет не очень красиво и не очень гибко в плане вывода. Поэтому сделаем следующим образом. У нас будет 4 колонки: Name, Health, Power, Seleted. Но в колонке Power мы выведем значение Power из хранилища и в GtkCellRendererSpin и в GtkCellRendererProgress.  
  
Нажмём правой клавишей по добавленной строке: column column и выберем там Add child Text item. В итоге у Вас должно получится следующее:



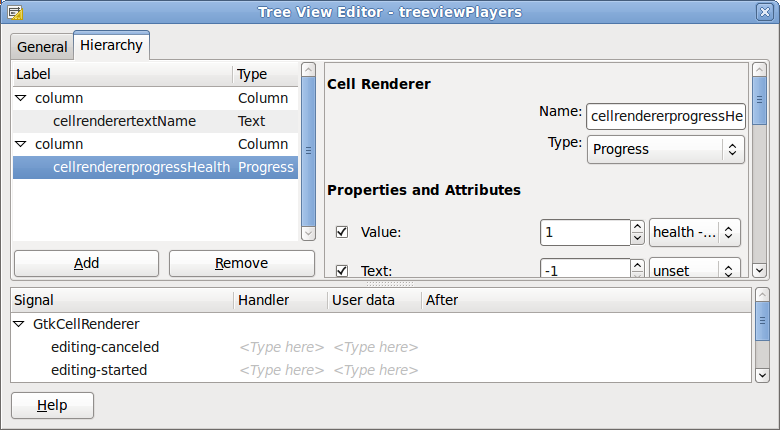
Тем самым мы добавили GtkCellRendererText — для отображения текстового содержимого. Начинаем изменять свойства, которые располагаются справа. В Name ставим название *cellrenderertextName*. Далее свойство Text. Там где у написано unset нажимаем на стрелочки и в выпадающем меню выбираем *name — gchararray*. Ноль означает, что привязана из хранилища (GtkListStore) колонка с номером 0, с хранящимися там данными (строка имени персонажа, которую мы задали в GtkListStore: *name* типа *gchararray*).



Прокручиваем список свойств вниз и ищем Background colour. Я предлагаю его задать жестко, т.е. для всех строк этого GtkCellRendererText будет одинаковый цвет фона. Для этого снимаем галочку и появится кнопка выбора цвета. Нажимаете и на неё и выбираете цвет, который Вам больше по душе. А вот ниже идёт свойство Foreground colour. Его я уже предлагаю брать из хранилища. Тем самым для каждого конкретной строки можно будет разное свойство цвета шрифта. Для этого нажимаем справо на unset и выбираем там *colortext — GdkColor*. И у Вас должна появится цифра 4. т.е. номер колонки из хранилища GtkListStore, где мы храним цвет. Так же необходимо сделать так, чтобы можно было редактировать имя прямо в Tree View. Для этого прокручиваем свойства еще ниже и ищем там Editable. Снимаем галочку и нажимаем кнопку, так чтобы появилось Yes. У Вас должно получится так:



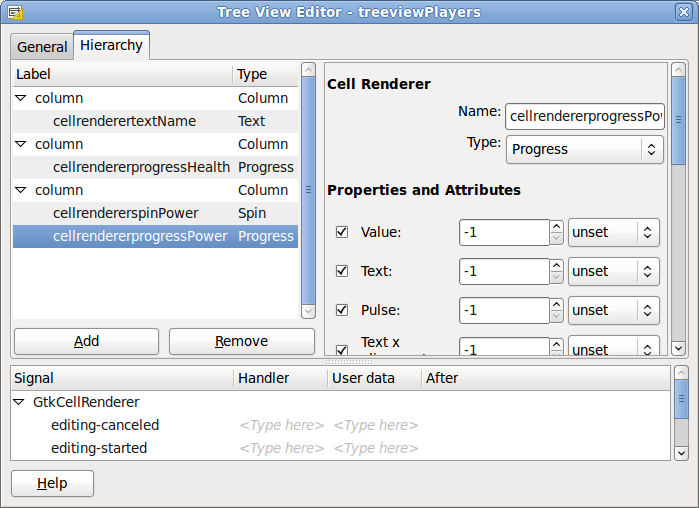
Я думаю стало уже более менее понятно в чём суть привязвки. Когда у свойства стоит галочка значит данные берутся из хранилища. По умолчанию у всех свойств стоит -1, т.е. не из какой колонки хранилища GtkListStore ничего не берётся. Наша задача указать номер колонки хранилища откуда будут браться значения для свойства. Если мы хотим сами жестко что-то задать — снимаем галочку и настраиваем как хотим. Но тогда уже эти настройки никак не будут зависеть от данных в хранилище GtkListStore и будут применены абсолютно ко всем строкам данного объекта представления.  
  
Далее создаём еще одну колонку. Нажимаем на нашу первую колонку, она выделется и нажимаем Add. Добавится еще одна колонка. Если бы у нас был выделен наш GtkCellRendererText, то добавился бы еще один объект представления в первую колонку. Glade порой немного неудобен. Но ничего страшного.  
  
Теперь добавим объект представления GtkCellRendererProgress, где будем отображать Health — здоровье нашего персонажа с помощью ProgressBar. Нажимаем правой клавишей по новой колонке и выбираем Add child Progress item. Устанавливаем свойство Name в *cellrendererprogressHealth*, в Value выбираем *health — gint*, появится цифра 1 — значит данные будут браться из первой колонки хранилища GtkListStore.



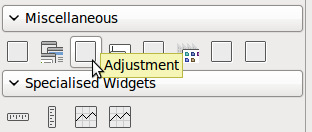
Устанавливаем свойство Background colour. Вы можете выбрать такой же цвет как и для GtkCellRendererText на предыдущем шаге (я именно так и сделал) а можете выбрать свой цвет. Может получится небольшой баг, когда вы выберете цвет на кнопке, он отобразится, а в строке код цвета будет #000000000000. Не расстраивайтесь. Цвет задан нормально, после того как вы закроете диалог редактирования этого Tree View и снова откроете, всё будет нормально. Небольшой глюк Glade.



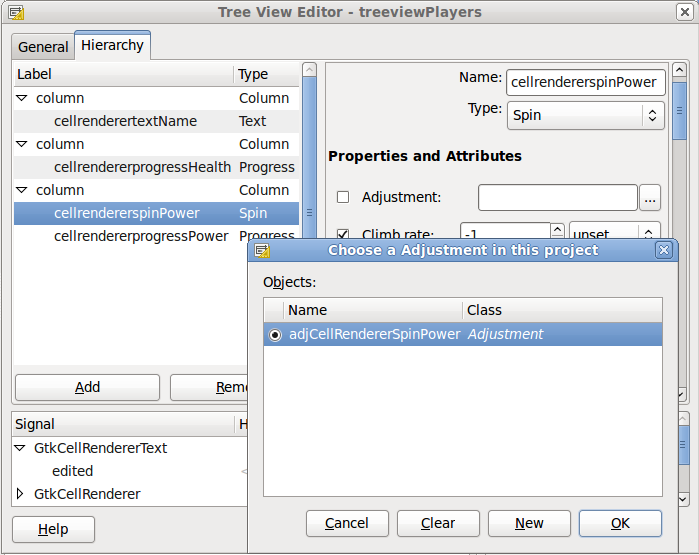
Теперь добавляем еще один column и в него вставляем GtkCellRendererSpin и GtkCellRendererProgress (точно так же как мы выше вставляли GtkCellRendererText). Для первого Add child Spin item, для второго Add child Progress item. Назовём их соответственно *cellrendererspinPower* и *cellrendererprogressPower* (свойство Name).



Теперь временно необходимо закрыть окно Tree View Editor и создать для cellrendererspinPower виджет GtkAdjustment под названием *adjCellRendererSpinPower*. На панели объектов выбираем Adjustment, как показано на рисунке ниже и он отобразиться в дереве виджетов в Objects.



В свойствах adjCellRendererSpinPower установим Maximum value: *100.00*, Page Size: *0.00*. Возвращаемся в режим редактирования нашего виджета GtkTreeView. Теперь необходимо определить adjCellRendererSpinPower для cellrendererspinPower. Снимаем галочку с Adjustment, нажимаем **...** и выбираем adjCellRendererSpinPower.



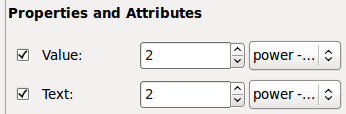
В Glade есть еще один нехороший баг. При новом открытии GladeXML файла с описанием интерфейса, который мы создаём сбрасывается вот этот самый флажок Adjustment. Не забывайте про это. Теперь устанавливаем другие свойства, начнём с Text. Выбираем из выпадаюшего списка *power — gint*, т.е. привязываем из хранилища колонка с номером 2 с данными, с данными о силе.



а так же Editable делаем *Yes*:



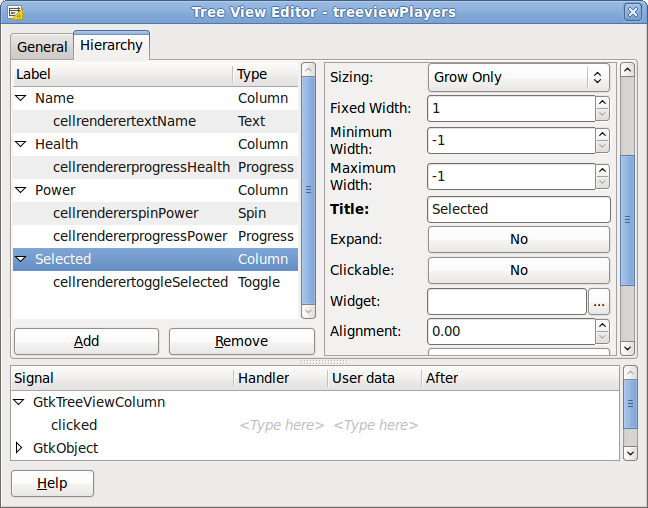
Еще необходимо установить, как и для прошлых объектов width: *60*; Cell background color: *#c0c0e5e5e7e7* (напоминаю, что у Вас может быть другой цвет) и Horizontal Padding: *5*. Теперь для cellrendererprogressPower устанавливаем Text (отображаемый текст на шкале) и Value (значение шкалы) тоже в *power — gint*.



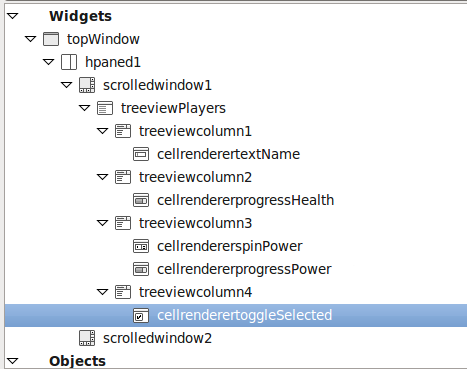
И конечно же установливаем, как и для прошлых объектов width: *100*; Cell background color: *#c0c0e5e5e7e7* и Horizontal Padding: *5*.  
  
Осталось добавить последний объект в GtkTreeView — cellrenderertoggleSelected — виджет типа Toggle (Add Toggle Item). Соотносим его свойство Toggle State с *selected gboolean*, с колонкой с номером 3 из хранилща с данными:



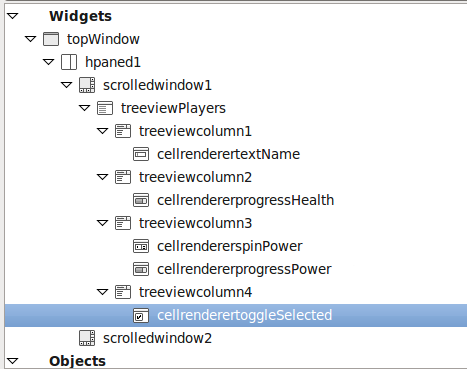
Так же устанавливаем Cell background color: *#c0c0e5e5e7e7*.  
  
На последок необходимо подписать колонки, т.е. установать для всех Column свойство Title. В соответствующие значения: *Name*, *Health*, *Power*, *Selected*:



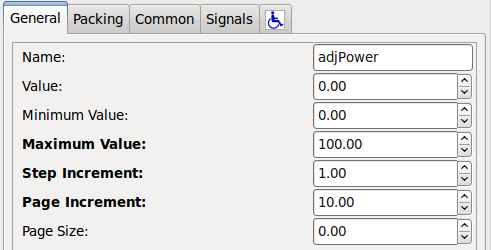
На этом настройка GtkTreeView почти завершена. Теперь дерево виджетов будет выглядеть следующим образом:



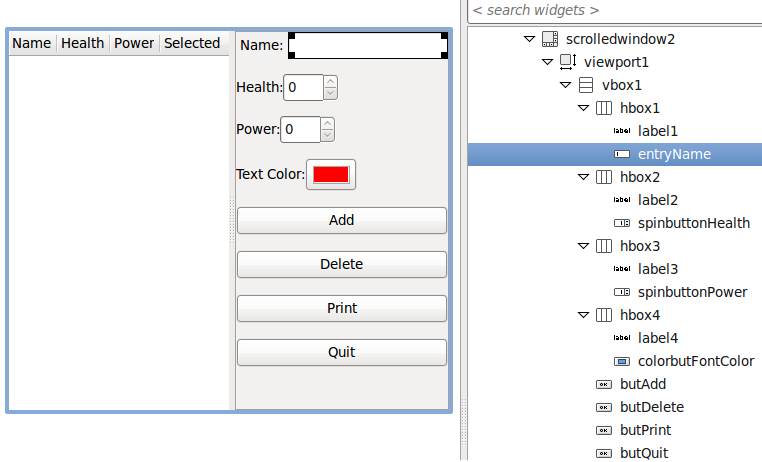
Далее необходимо прописать обработчики сигналов (событий) для некоторых виджетов GtkTreeView и для него самого. Нажимаем на виджет cellrenderertextName и переходим во вкладку Signals (сигналы), ищем там событие (я буду всё-таки сигналы называть как события) *edited* — редактирование текстового поля. С помощью выпадающего списка или в ручную пишем название обработчика этого события *on\_cellrenderertextName\_edited*, как показано на рисунке ниже:



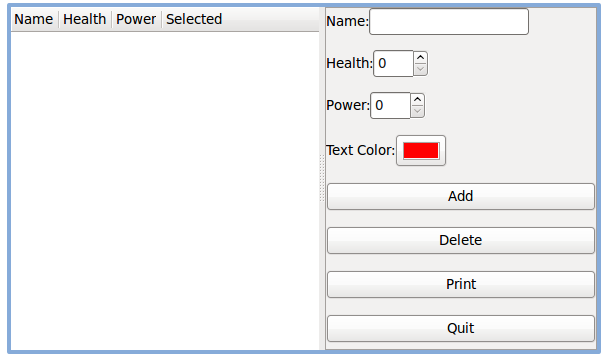
Точно так же делаем для виджета cellrendererspinPower для события edited обработчик *on\_cellrendererspinPower\_edited* — изменение значения с помощью «карусели». Для виджета cellrenderertoggleSelected ищем событие toggled — изменение состояния переключателя и выбираем название обработчика *on\_cellrenderertoggleSelected\_toggled*. И теперь нажимаем на сам наш виджет treeviewPlayers и ищем там во вкладке Signals row\_activated. Пишем сами или выбираем название обработчика on\_treeviewPlayers\_row\_activated.  
Хотя Вы в принципе можете как Вам больше нравится называть обработчики событий.Главное их так же как и в Glade указать далее в коде.  
  
Теперь необходимо добавить небольшую диалоговую панель, с помощью которой можно будет быстро добавлять новых персожей и вызывать другие манипуляции. Я не буду подробно описывать процесс создания этого меню, т.к. цель поста рассказать про GtkTreeView и GtkListStore, просто вкратце скажу основые шаги и покажу, что в итоге должно получиться.  
  
Сначала для scrolledwindow1 установим оба свойства Resize и Shrink в *Yes* (вкладка Packing) и Width request в *370* (вкладка Common). Для scrolledwindow2 свойство Resize в *No*, а Shrink в *Yes*, а Width request в *250*. Далее добавим виджет GtkViewport в scrolledwindow2, а в него уже виджет GtkVBox(вертикальный контейнер), состоящий из 8 ячеек. Первые четыре будут представлять собой виджеты GtkHBox (горизонтальный контейнер), состоящие из GtkLabel (подписи) и контролов: entryName (GtkEntry), spinbuttonHealth (GtkSpinButton), spinbuttonPower (GtkSpinButton) и colorbutFontColor (GtkColorButton). Поясняю: необходимо добавить 4 GtkHBox, состоящих из 2 двух ячеек: одна для подписи (левая), другая для контрола (правая), перечисленных ранее.  
Четыре нижних это кнопки: butAdd, butDelete, butPrint, butQuit. Для всех виджетов внутри GtkVBox рекомендую установить во вкладке Packing свойства Expand и Fill в *No* (об этих свойствах рассказывается в моём предыдущем [посте](http://habrahabr.ru/blogs/development/107403/)). Так же виджеты spinbuttonHealth и spinbuttonPower необходимо связать с соответствующими виджетами GtkAdjustment. Я их назвал adjHealth и adjPower. И установился соответствующие настройки:



Связывание с spinbuttonHealth и spinbuttonPower происходит посредством выбора в их свойствах Adjustment созданных *adjHealth* и *adjPower*.  
  
Всё, что осталось сделать в Glade — это определить обработчики для кнопок: butAdd, butDelete, butPrint, butQuit. Нажимаем на каждый виджет и во вкладке Event для свойства clicked выбираем *on\_butAdd\_clicked*, *on\_butDelete\_clicked*, *on\_butPrint\_clicked*, *on\_butQuit\_clicked*.  
  
В итоге должно получиться следующее:



А в целом всё окно нашего тренировочного приложения выглядит так:



Теперь переходим к написанию кода. Весь код будет содержаться в файле Example1.cpp, хотя Вы можете назвать его как Вам больше нравится.

#**include** <stdlib.h>

#**include** <gtk/gtk.h>

#**define** UI\_FILE "mainForm.glade"

Структура для работы с хранилищем, о которой было сказано в начале:

**enum**

{

NAME = 0,

HEALTH,

POWER,

SELECTED,

FONT\_COLOR

};

Протопипы обработчиков событий, названия должно быть такими, как мы указали во вкладе Signals для виджетов в редакторе Glade. Т.к. используется C++ компилятор, то указываем, что они должны связываться согласно порядку связывания в языке С.

**extern** "C" **void** **on\_butQuit\_clicked**(GtkWidget \*TopWindow, gpointer data);

**extern** "C" **void** **on\_butAdd\_clicked**(GtkWidget \*window, gpointer data);

**extern** "C" **void** **on\_butDelete\_clicked** (GtkButton \*remove, gpointer data);

**extern** "C" **void** **on\_butPrint\_clicked**(GtkWidget \*button, gpointer data);

**extern** "C" **void** **on\_cellrendererspinPower\_edited**(GtkCellRendererText \*render, gchar \*path, gchar \*new\_text, gpointer data);

**extern** "C" **void** **on\_cellrenderertoggleSelected\_toggled**(GtkCellRendererToggle \*render, gchar \*path, gpointer data);

**extern** "C" **void** **on\_treeviewPlayers\_row\_activated**(GtkTreeView \*treeview, GtkTreePath \*path, GtkTreeViewColumn \*col, gpointer data);

**extern** "C" **void** **on\_cellrenderertextName\_edited**(GtkCellRendererText \*renderer, gchar \*path, gchar \*new\_text, gpointer data);

Структура, содержащая указатели на виджеты, которые был созданы в Glade и будут получены из GladeXML в функции main.

**struct** **MainWindowObjects**

{

GtkWindow \*topWindow;

GtkTreeView \*treeviewPlayers;

GtkListStore \*liststorePlayers;

GtkEntry \*entryName;

GtkAdjustment \*adjHealth;

GtkAdjustment \*adjPower;

GtkColorButton \*colorbutFontColor;

} mainWindowObjects;

В main получаем виджеты из GladeXML-формата. С помощью gtk\_tree\_selection\_set\_mode устанавливаем возможность выделения в GtkTreeView нескольких строк. Связываем наши обработчики событий используя функцию gtk\_builder\_connect\_signals и заодно передаём указатели на виджеты с помощью ссылки mainWindowObjects. Тем самым, как вы увидите дальше с помощью указателя gpointer data можно будет обращаться ко всем виджетам из функции обработчика.

**int** **main**(**int** argc, **char**\*\* argv)

{

GtkBuilder \*builder;

GError \*error = NULL;

gtk\_init( &argc, &argv );

builder = gtk\_builder\_new();

**if**( ! gtk\_builder\_add\_from\_file( builder, UI\_FILE, &error ) )

{

g\_warning( "%s\n", error->message );

g\_free( error );

**return**( 1 );

}

mainWindowObjects.topWindow = GTK\_WINDOW(gtk\_builder\_get\_object(builder, "topWindow"));

mainWindowObjects.treeviewPlayers = GTK\_TREE\_VIEW( gtk\_builder\_get\_object( builder, "treeviewPlayers" ) );

mainWindowObjects.liststorePlayers = GTK\_LIST\_STORE( gtk\_builder\_get\_object(builder, "liststorePlayers") );

mainWindowObjects.entryName = GTK\_ENTRY( gtk\_builder\_get\_object(builder, "entryName") );

mainWindowObjects.adjHealth = GTK\_ADJUSTMENT( gtk\_builder\_get\_object(builder, "adjHealth") );

mainWindowObjects.adjPower = GTK\_ADJUSTMENT( gtk\_builder\_get\_object(builder, "adjPower") );

mainWindowObjects.colorbutFontColor = GTK\_COLOR\_BUTTON( gtk\_builder\_get\_object(builder, "colorbutFontColor") );

GtkTreeSelection \*selection;

selection = gtk\_tree\_view\_get\_selection( GTK\_TREE\_VIEW(mainWindowObjects.treeviewPlayers) );

gtk\_tree\_selection\_set\_mode( selection, GTK\_SELECTION\_MULTIPLE );

gtk\_builder\_connect\_signals (builder, &mainWindowObjects);

g\_object\_unref( G\_OBJECT( builder ) );

gtk\_widget\_show\_all ( GTK\_WIDGET (mainWindowObjects.topWindow) );

gtk\_main ();

}

Реализация обработчика добавления нового персонажа. gtk\_list\_store\_append — добавляет новую строку в модель (хранилище) и возвращает iter — ссылку на неё (или еще это называют итератором, указывающим на строку). А функция gtk\_list\_store\_set устанавливает значения ячеек в строке, на которую указывает iter. Значения указываются с помощью перечисления: NAME, HEALTH и т.д. Признак окончания это -1.

**void** **on\_butAdd\_clicked**(GtkWidget \*button, gpointer data)

{

MainWindowObjects\* mwo = **static\_cast**<MainWindowObjects\*>( data );

GtkTreeIter iter;

GdkColor color;

gtk\_color\_button\_get\_color( mwo->colorbutFontColor, &color );

gtk\_list\_store\_append(GTK\_LIST\_STORE( mwo->liststorePlayers ), &iter);

gtk\_list\_store\_set(GTK\_LIST\_STORE( mwo->liststorePlayers ), &iter,

NAME, gtk\_entry\_get\_text(mwo->entryName),

HEALTH, **static\_cast**<**int**>( gtk\_adjustment\_get\_value(mwo->adjHealth) ),

POWER, **static\_cast**<**int**>( gtk\_adjustment\_get\_value(mwo->adjPower) ),

SELECTED, false,

FONT\_COLOR, &color,

-1 );

}

Реализация обработчика события редактирования ячейки с именем. С помощью функции gtk\_tree\_view\_get\_model получаем модель (хранилище), связанную с нашим GtkTreeView. Используя функцию gtk\_tree\_model\_get\_iter\_from\_string получаем ссылку на строку из пути path (который находится в сигнатуре функции, для данного представления хранилища этот путь — просто номер строки, которую изменили), а дальше с помощью уже знакомой нам gtk\_list\_store\_set устанавливаем новое значение, которое находится в new\_text. Вообще сигнатура данного обработчика (как и остальных обработчиков) взята из официальной [документации](http://developer.gnome.org/gtk/2.24/GtkCellRendererText.html) по Gtk+.  
Т.е. для каждого виджета можно найти там сигнатуру события (или как там пишут signal — сигнатуру сигнала).

**void** **on\_cellrenderertextName\_edited**(GtkCellRendererText \*renderer, gchar \*path, gchar \*new\_text, gpointer data)

{

MainWindowObjects\* mwo = **static\_cast**<MainWindowObjects\*>( data );

**if** ( g\_ascii\_strcasecmp(new\_text, "") != 0 )

{

GtkTreeIter iter;

GtkTreeModel \*model;

model = gtk\_tree\_view\_get\_model (mwo->treeviewPlayers);

**if** (gtk\_tree\_model\_get\_iter\_from\_string(model, &iter, path) )

gtk\_list\_store\_set(GTK\_LIST\_STORE (model), &iter, NAME, new\_text, -1 );

}

}

Реализация обработчика события редактирвоания с помощью кнопки «карусель». Оно почти аналогично описанному выше. За исключением того, что нужно новое значение преобразовать в int. Для простоты я использовал старую Си-шную функцию atoi, хотя можно было реализовать это, используя stringstream.

**void** **on\_cellrendererspinPower\_edited**(GtkCellRendererText \*renderer, gchar \*path, gchar \*new\_text, gpointer data)

{

MainWindowObjects\* mwo = **static\_cast**<MainWindowObjects\*>( data );

**if** ( g\_ascii\_strcasecmp(new\_text, "") != 0 )

{

GtkTreeIter iter;

GtkTreeModel \*model;

model = gtk\_tree\_view\_get\_model (mwo->treeviewPlayers);

**if** (gtk\_tree\_model\_get\_iter\_from\_string(model, &iter, path) )

gtk\_list\_store\_set(GTK\_LIST\_STORE (model), &iter, POWER, atoi(new\_text), -1 );

}

}

Реализация обработчика нажатия по ячейке с переключателем (флагом) — 4я колонка. Получаем с помощью gtk\_tree\_model\_get текущение состояние переключателя и устанавливаем противоложное: !selected

**void** **on\_cellrenderertoggleSelected\_toggled**(GtkCellRendererToggle \*render, gchar \*path, gpointer data)

{

MainWindowObjects\* mwo = **static\_cast**<MainWindowObjects\*>( data );

GtkTreeIter iter;

GtkTreeModel \*model;

gboolean selected;

model = gtk\_tree\_view\_get\_model ( mwo->treeviewPlayers );

**if** (gtk\_tree\_model\_get\_iter\_from\_string(model, &iter, path) )

{

gtk\_tree\_model\_get( model, &iter, 3, &selected, -1 );

gtk\_list\_store\_set( GTK\_LIST\_STORE (model), &iter, SELECTED, !selected, -1 );

}

}

Реализация метода печати выделенных строк. С помощью функции gtk\_tree\_model\_get\_iter\_first получаем ссылку на первую строку и заодно проверяем: не пусто ли хранилище (флаг reader). С помощью метода gtk\_tree\_model\_get получаем интересующие нас значения ячеек для текущей строки, на которую «смотрит» iter. К следующей строке переходим с помощью gtk\_tree\_model\_iter\_next.

**void** **on\_butPrint\_clicked**(GtkWidget \*button, gpointer data)

{

MainWindowObjects\* mwo = **static\_cast**<MainWindowObjects\*>( data );

GtkTreeIter iter;

gboolean reader = gtk\_tree\_model\_get\_iter\_first(GTK\_TREE\_MODEL( mwo->liststorePlayers ), &iter);

g\_print( "Selected rows:\n" );

**while** ( reader )

{

gboolean selected;

gtk\_tree\_model\_get (GTK\_TREE\_MODEL( mwo->liststorePlayers ), &iter,

SELECTED, &selected, -1);

**if** ( selected )

{

gchar\* name;

gint health;

gint power;

gtk\_tree\_model\_get (GTK\_TREE\_MODEL( mwo->liststorePlayers ), &iter,

NAME, &name,

HEALTH, &health,

POWER, &power,

-1);

g\_print( "%s %d %d\n", name, health, power );

}

reader = gtk\_tree\_model\_iter\_next(GTK\_TREE\_MODEL( mwo->liststorePlayers ), &iter);

}

}

Реализация метода, обрабатывающего двойной щелчок по строке в GtkTreeView. С помощью функции gtk\_tree\_model\_get\_iter получаем iter из path, а потом получаем интересующие нас данные с помощью gtk\_tree\_model\_get и выводим их в консоль.

**void** **on\_treeviewPlayers\_row\_activated**(GtkTreeView \*treeview, GtkTreePath \*path, GtkTreeViewColumn \*col, gpointer data)

{

GtkTreeModel \*model;

GtkTreeIter iter;

model = gtk\_tree\_view\_get\_model( treeview );

**if** ( gtk\_tree\_model\_get\_iter(model, &iter, path) )

{

gchar\* name;

gint health;

gint power;

gtk\_tree\_model\_get(model, &iter,

NAME, &name,

HEALTH, &health,

POWER, &power,

-1);

g\_print( "Current row: %s %d %d\n", name, health, power);

}

}

Данная функция является вспомогательной, для удаления выделенных строк в GtkTreeView. gtk\_list\_store\_remove — удаляет данную строку из списка, на которую ссылается iter. Перед этим извлекаем path с помощью gtk\_tree\_row\_reference\_get\_path, а потом преобразуем path в iter с помощью gtk\_tree\_model\_get\_iter.

**static** **void** **remove\_row** (GtkTreeRowReference \*ref, GtkTreeModel \*model)

{

GtkTreeIter iter;

GtkTreePath \*path;

path = gtk\_tree\_row\_reference\_get\_path (ref);

gtk\_tree\_model\_get\_iter (model, &iter, path);

gtk\_list\_store\_remove (GTK\_LIST\_STORE (model), &iter);

}

С помощью функции gtk\_tree\_selection\_get\_selected\_rows получаем список выделенных строк, потом с помощью цикла while проходим по каждой выделенной строке и создаем ссылку на основе пути с помощью функции gtk\_tree\_row\_reference\_new и добавляем эту ссылку в список references с помощью g\_list\_prepend.  
Вызываем функцию remove\_row для каждой ссылки, с помощью g\_list\_foreach. И в конце, аналогично с помощью g\_list\_foreach освобождаем наши списки.

**void** **on\_butDelete\_clicked** (GtkButton \*remove, gpointer data)

{

MainWindowObjects\* mwo = **static\_cast**<MainWindowObjects\*>( data );

GtkTreeSelection \*selection;

GtkTreeRowReference \*ref;

GtkTreeModel \*model;

GList \*rows, \*ptr, \*references = NULL;

selection = gtk\_tree\_view\_get\_selection ( mwo->treeviewPlayers );

model = gtk\_tree\_view\_get\_model ( mwo->treeviewPlayers );

rows = gtk\_tree\_selection\_get\_selected\_rows (selection, &model);

ptr = rows;

**while** (ptr != NULL)

{

ref = gtk\_tree\_row\_reference\_new (model, (GtkTreePath\*) ptr->data);

references = g\_list\_prepend (references, gtk\_tree\_row\_reference\_copy (ref));

gtk\_tree\_row\_reference\_free (ref);

ptr = ptr->next;

}

g\_list\_foreach ( references, (GFunc) remove\_row, model );

g\_list\_foreach ( references, (GFunc) gtk\_tree\_row\_reference\_free, NULL );

g\_list\_foreach ( rows, (GFunc) gtk\_tree\_path\_free, NULL );

g\_list\_free ( references );

g\_list\_free ( rows );

}

Реализация обработчика закрытия GTK+ приложения

**void** **on\_butQuit\_clicked**(GtkWidget \*window, gpointer data)

{

gtk\_main\_quit();

}

Для компиляции необходимо создать makefile:

CC=g++

LDLIBS=`pkg-config --libs gtk+-2.0 gmodule-2.0`

CFLAGS=-Wall -g `pkg-config --cflags gtk+-2.0 gmodule-2.0`

example1: example1.o

$(CC) $(LDLIBS) example1.o -o example1

example1.o: example1.cpp

$(CC) $(CFLAGS) -c example1.cpp

clean:

rm -f example1

rm -f \*.o

Теперь можно скомпилировать данный пример и попробовать «поиграться» с тем, что мы сделали. Советую запускать из консоли дистрибутива вашего дистрибутива Linux. Попробуйте добавить несколько персонажей, выбирая разные цвета их имени. Попробуйте выделить в 4ой колонке несколько строк и нажать Print, тогда в консоле напечатаются выделенные. Попробуйте изменить значение Power с помощью «карусели» (двойной щелчок по ней, левее Progress bar'a). Попробуйте с помощью зажатой клавишы Ctrl или Shift выделить несколько добавленных строк и удалить их, нажав Delete.  
  
Данный пример можно скачать [отсюда](https://bitbucket.org/goran_src/example1/src).  
  
Компиляция происходит командой:

$ make

Запуск:

$ ./example1

Я постарался максимально подробно описать многие вещи (некоторые может быть даже слишком), но всё это сделано для более быстрого понимания данной темы.

**Теги:**[gtk](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bgtk%5D)[gtk+](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bgtk%2B%5D)[glade](https://habr.com/ru/search/?target_type=posts&order=relevance&q=%5Bglade%5D)

**Хабы:**[Программирование](https://habr.com/ru/hub/programming/)