15th April 2009

GtkComboBox widget - part 2

As promised, I'm back with second part of my GtkComboBox widget tutorial.

Today, we'll be learning about "complex" API. The main thing to keep in mind when dealing with GtkComboBox using complex API is the fact that data is separated from it's representation, meaning that one object holds data (usually GtkTreeStore or GtkListStore), the other object renders it to the screen (GtkCellRenderer) and third object manipulates it (GtkComboBox in our case). This concept (also known as Model/View/Controller-design or MVC) is also used in GtkTreeView and GtkIconView widgets. GtkCellLayout interface provides the means to achieve all this.

Contents of this tutorial:

Complex API

Let's start creating our combo box. We'll rewrite sample application from part 1 using the complex API.

First we need to create data object and fill it with some information. Since we're creating simple text list, our data object will be GtkListStore with only one text column. I won't go into details about creating and updating data store, since excellent description already exists in GtkTreeView tutorial (Chapter 3) [http://scentric.net/tutorial/sec-treemodels.html] . Read it first if you're not familiar with GtkTreeStore, GtkListStore; GtkTreeModel; GtkTreeIter and GtkTreePath.

```
/* Create data store ... */
store = gtk_list_store_new( 1, G_TYPE_STRING );

/* ... and fill it with some information. */
gtk_list_store_append( store, &iter );
gtk_list_store_set( store, &iter, 0, "Hello World once", -1 );
gtk_list_store_prepend( store, &iter );
gtk_list_store_set( store, &iter, 0, "Hello World twice", -1 );
gtk_list_store_insert( store, &iter, 1 );
gtk_list_store_set( store, &iter, 0, "Hello World last time", -1 );
```

Now that we have our data safely stored inside GtkListStore object (model part of MVC), we'll create combo box and connect it to the store.

```
/* Create combo box with store as data source. */
combo = gtk_combo_box_new_with_model( GTK_TREE_MODEL( store ) );

/* Remove our reference from store to avoid memory leak. */
g_object_unref( G_OBJECT( store ) );
```

We now have model and controller parts of MVC, so all there is left to do is to create view. We want to display strings in our combo box, so we'll use GtkCellRendererText as view. After we create our renderer, we need to pack it inside our combo box and connect it to data source.

```
/* Create cell renderer. */
cell = gtk_cell_renderer_text_new();

/* Pack it to the combo box. */
gtk_cell_layout_pack_start( GTK_CELL_LAYOUT( combo ), cell, TRUE );

/* Connect renderer to data source */
gtk_cell_layout_set_attributes( GTK_CELL_LAYOUT( combo ), cell, "text", 0, NULL );
```

In this code snippet we finally came across the mysterious GtkCellLayout interface. We used functions from it to add cell renderer to combo box and connect data with renderer. What exactly the last line of code does? We tell cell renderer that values for it's "text" property can be found in first column of the underlying data store. And that's it! We now have working combo box. But how to retrieve data from it like we did in simple example?

The process of retrieving data is composed from three steps:

- Getting active iter from combo box.
- Getting model from combo box.
- Getting data from model.

This might seem a bit complicated at first, but if you ever used GtkTreeView, you should feel right at home. This is how these steps look in code:

```
/* Obtain currently selected item from combo box.
 * If nothing is selected, do nothing. */
if( gtk_combo_box_get_active_iter( combo, &iter ) )
{
    /* Obtain data model from combo box. */
    model = gtk_combo_box_get_model( combo );

    /* Obtain string from model. */
    gtk_tree_model_get( model, &iter, 0, &string, -1 );
}
```

And the last thing that we have to implement to imitate previous sample application is removing currently selected item. This operation is very similar to retrieving data from combo box and is done in three steps (again;):

- Getting active iter from combo box.
- Getting model from combo box.
- Removing item from model.

And the code for these steps:

```
/* Obtain currently selected item form combo box.
 * If nothing is selected, do nothing. */
if( gtk_combo_box_get_active_iter( combo, &iter ) )
{
    /* Obtain model from combo box. */
    store = GTK_LIST_STORE( gtk_combo_box_get_model( combo ) );

    /* Remove item. */
    gtk_list_store_remove( store, &iter );
}
```

And the last thing we need to do is to test our application. Here is complete source code (highlighted parts are the ones that have been changed from simple to complex version):

```
/*
 * Test me with:
 *  gcc -o combo1 combo1.c $(pkg-config --cflags --libs gtk+-2.0) && ./combo1
 */

#include <gtk/gtk.h>

/* This function gets called when currently selected item changes. */
static void
cb_changed( GtkComboBox *combo,
            gpointer     data )
{
    GtkTreeIter   iter;
    gchar        *string = NULL;
    GtkTreeModel *model;

    /* Obtain currently selected item from combo box.
     * If nothing is selected, do nothing. */
    if( gtk_combo_box_get_active_iter( combo, &iter ) )
    {
        /* Obtain data model from combo box. */
        model = gtk_combo_box_get_model( combo );

        /* Obtain string from model. */
        gtk_tree_model_get( model, &iter, 0, &string, -1 );
    }

    /* Print string to the console - if string is NULL, print NULL. */
    g_print( "Selected (complex): >> %s <<\n", ( string ? string : "NULL" ) );

    /* Free string (if not NULL). */
    if( string )
        g_free( string );
}

/* This function deletes currently selected item from combo box. */
static void
cb_delete( GtkButton   *button,
           GtkComboBox *combo )
{
    GtkTreeIter   iter;
    GtkListStore *store;

    /* Obtain currently selected item form combo box.
     * If nothing is selected, do nothing. */
    if( gtk_combo_box_get_active_iter( combo, &iter ) )
    {
        /* Obtain model from combo box. */
```

```
        store = GTK_LIST_STORE( gtk_combo_box_get_model( combo ) );

        /* Remove item. */
        gtk_list_store_remove( store, &iter );
    }
}


int
main( int    argc,
      char **argv )
{
    GtkWidget       *window;
    GtkWidget       *vbox;
    GtkWidget       *frame;
    GtkWidget       *combo;
    GtkWidget       *button;

    GtkListStore    *store;
    GtkTreeIter      iter;
    GtkCellRenderer *cell;

    /* Initialization */
    gtk_init( &argc, &argv );

    /* Create main window */
    window = gtk_window_new( GTK_WINDOW_TOPLEVEL );
    g_signal_connect( G_OBJECT( window ), "destroy",
                      G_CALLBACK( gtk_main_quit ), NULL );
    gtk_container_set_border_width( GTK_CONTAINER( window ), 10 );

    /* Create vbox */
    vbox = gtk_vbox_new( FALSE, 6 );
    gtk_container_add( GTK_CONTAINER( window ), vbox );

    /* Create frame */
    frame = gtk_frame_new( "Complex API" );
    gtk_box_pack_start( GTK_BOX( vbox ), frame, FALSE, FALSE, 0 );

    /* Create data store ... */
    store = gtk_list_store_new( 1, G_TYPE_STRING );

    /* ... and fill it with some information. */
    gtk_list_store_append( store, &iter );
    gtk_list_store_set( store, &iter, 0, "Hello World once", -1 );
    gtk_list_store_prepend( store, &iter );
    gtk_list_store_set( store, &iter, 0, "Hello World twice", -1 );
    gtk_list_store_insert( store, &iter, 1 );
    gtk_list_store_set( store, &iter, 0, "Hello World last time", -1 );

    /* Create combo box with store as data source. */
```

```
    combo = gtk_combo_box_new_with_model( GTK_TREE_MODEL( store ) );
    gtk_container_add( GTK_CONTAINER( frame ), combo );

    /* Remove our reference from store to avoid memory leak. */
    g_object_unref( G_OBJECT( store ) );

    /* Create cell renderer. */
    cell = gtk_cell_renderer_text_new();

    /* Pack it into the combo box. */
    gtk_cell_layout_pack_start( GTK_CELL_LAYOUT( combo ), cell, TRUE );

    /* Connect renderer to data source. */
    gtk_cell_layout_set_attributes( GTK_CELL_LAYOUT( combo ), cell, "text", 0, NULL );

    /* Connect signal to our handler function. */
    g_signal_connect( G_OBJECT( combo ), "changed",
                      G_CALLBACK( cb_changed ), NULL );

    /* Add button that, when clicked, deletes currently selected entry. */
    button = gtk_button_new_with_mnemonic( "_Delete selected item" );
    gtk_box_pack_start( GTK_BOX( vbox ), button, FALSE, FALSE, 0 );
    g_signal_connect( G_OBJECT( button ), "clicked",
                      G_CALLBACK( cb_delete ), GTK_COMBO_BOX( combo ) );

    /* Show our application and start main loop. */
    gtk_widget_show_all( window );
    gtk_main();

    return( 0 );
}
```

And that will conclude our lesson for today. Join me next time in part three, when we'll be making salad from images, labels and progress bars, all served in GtkComboBox bowl;)

Bye

Posted 15th April 2009 by Tadej Borovšak

Labels: GTK+, GtkComboBox, tutorial

4 View comments

Miguel Ángel 11 October, 2009 11:19

Great. Very useful. I'll bookmark your blog. Thank you. Keep on the good work.

Reply

Николай 16 November, 2010 22:09

The tutorial is fine and clear. It helped me to go ahead with my learning Gtk+ programming. I found out how GtkComboBox can be properly used in practice. Thanks.

Reply

Andy Shevchenko 08 June, 2011 17:57

if (smth) free(smth) is overkill. free() is NULL safe.

Reply

Anonymous 21 February, 2012 12:13

Great. The tutorial is fine and clear. Thank you Tadej.

Reply

Enter your comment...

Comment as: Google Accou ▼

Publish Preview