

# **Programming with Clutter**

**Murray Cumming**

## **Programming with Clutter**

by Murray Cumming

Copyright © 2007 Openismus GmbH

We very much appreciate any reports of inaccuracies or other errors in this document. Contributions are also most welcome. Post your suggestions, critiques or addenda to the team (<mailto:murrayc@openismus.com>).

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. You may obtain a copy of the GNU Free Documentation License from the Free Software Foundation by visiting their Web site or by writing to: Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1. This book.....	1
1.2. Clutter.....	1
<b>2. Installation.....</b>	<b>3</b>
2.1. Prebuilt Packages .....	3
2.2. Installing From Source .....	3
2.2.1. Dependencies.....	3
<b>3. The Stage .....</b>	<b>4</b>
3.1. Stage Basics .....	4
3.1.1. Example .....	4
3.2. Stage Widget .....	6
3.2.1. Example .....	6
<b>4. Actors.....</b>	<b>10</b>
4.1. Actor Basics .....	10
4.1.1. Example .....	10
4.2. Transformations .....	12
4.2.1. Scaling .....	12
4.2.2. Rotation .....	12
4.2.3. Clipping .....	13
4.2.4. Movement.....	13
4.2.5. Example .....	13
4.3. Containers .....	15
4.3.1. Example .....	16
4.4. Events.....	17
4.4.1. Example .....	18
<b>5. Timelines.....</b>	<b>23</b>
5.1. Using Timelines .....	23
5.2. Example .....	23
5.3. Grouping TimeLines in a Score .....	26
5.4. Example .....	26
<b>6. Effects.....</b>	<b>30</b>
6.1. Using Effects .....	30
6.2. Example .....	31
<b>7. Behaviours .....</b>	<b>34</b>
7.1. Using Behaviours .....	34
7.2. Example .....	36
<b>8. Full Example .....</b>	<b>40</b>
<b>A. Implementing Actors .....</b>	<b>51</b>
A.1. Implementing Simple Actors .....	51
A.2. Example.....	52
A.3. Implementing Container Actors.....	59
A.3.1. ClutterActor virtual functions to implement .....	60
A.3.2. ClutterContainer virtual functions to implement.....	60

A.4. Example.....	61
<b>B. Implementing Scrolling in a Window-like Actor .....</b>	<b>72</b>
B.1. The Technique .....	72
B.2. Example.....	72
<b>C. Implementing Text Editing .....</b>	<b>84</b>
C.1. The Technique .....	84
C.2. Example.....	84
<b>9. Contributing.....</b>	<b>109</b>

# List of Figures

- 3-1. Stage .....4
- 3-2. Stage Widget .....6
- 4-1. Actor .....11
- 4-2. Actor .....13
- 4-3. Group.....16
- 4-4. Actor Events .....19
- 5-1. Timeline.....23
- 5-2. Score .....26
- 6-1. Graphic representation of some alpha functions. ....30
- 6-2. Behaviour .....31
- 7-1. Effects of alpha functions on a path. ....34
- 7-2. Behaviour .....36
- 8-1. Full Example .....40
- A-1. Behaviour .....52
- A-2. Behaviour .....61
- B-1. Scrolling Container.....72
- C-1. Multiline Text Entry.....84

# Chapter 1. Introduction

## 1.1. This book

This book assumes a good understanding of C, and how to create C programs.

This book attempts to explain key Clutter concepts and introduce some of the more commonly used user interface elements ("actors"). For full API information you should follow the links into the reference documentation. This document covers the API in Clutter version 0.6.

Each chapter contains very simple examples. These are meant to show the use of the API rather than show an impressive visual result. However, the full example should give some idea of what can be achieved with Clutter

The Clutter platform uses techniques found in the GTK+ (<http://www.gtk.org>) platform, so you will sometimes wish to refer to the GTK+ documentation.

We would very much like to hear of any problems you have learning Clutter with this document, and would appreciate input regarding improvements. Please see the Contributing section for further information.

## 1.2. Clutter

Clutter is a C programming API that allows you to create simple but visually appealing and involving user interfaces. It offers a variety of objects (actors) which can be placed on a canvas (stage) and manipulated by the application or the user. It is therefore a "retained mode" graphics API. Unlike traditional 2D canvas APIs, Clutter allows these actors to move partly in the Z dimension.

This concept simplifies the creation of 3D interfaces compared to direct use of OpenGL or other 3D drawing APIs. For instance, it restricts the user interaction to the 2D plane facing the user, which is appropriate for today's devices allowing interaction only with a 2D plane such as a touchscreen. In addition, your application does not need to provide visual context to show the user which objects are, for instance, small rather than far away.

In addition, Clutter provides timeline and behavior abstractions which simplify animation by allowing you to associate actor properties (such as position, rotation, or opacity) with callback functions, including pre-defined functions of time such as sine waves.

Clutter uses the popular OpenGL 3D graphics API on regular desktop PCs, allowing it access to hardware acceleration. On handheld devices it can use OpenGL ES, a subset of the OpenGL API aimed at embedded devices. So, where necessary, you may also use OpenGL or OpenGL ES directly.

In the next few chapters you will learn how to place actors on the stage, how to set their properties, how to change their properties (including their position) over time by using timelines and behaviours, and how to do all this in response to user interaction.

# Chapter 2. Installation

## 2.1. Prebuilt Packages

Clutter packages are probably available from your Linux distribution. For instance, on Ubuntu Linux or Debian you can install the `libclutter-0.6-dev` package.

## 2.2. Installing From Source

After you've installed all of the dependencies, download the Clutter source code, unpack it, and change to the newly created directory. Clutter can be built and installed with the following sequence of commands:

```
# ./configure
# make
# make install
```

The `configure` script will check to make sure all of the required dependencies are already installed. If you are missing any dependencies it will exit and display an error.

By default, Clutter will be installed under the `/usr/local` directory.

If you want to help develop Clutter or experiment with new features, you can also install Clutter from SVN. Details are available at the Clutter web site (<http://www.clutter-project.org/>).

### 2.2.1. Dependencies

Before attempting to install Clutter, you should first install these other packages:

- GTK+
- libgl (Mesa)

These dependencies have their own dependencies, including the following applications and libraries:

- pkg-config
- glib
- ATK
- Pango

# Chapter 3. The Stage

## 3.1. Stage Basics

Each Clutter application contains at least one `ClutterStage`. Actually, Clutter currently allows only one stage per application, but this limitation might be removed in future. This stage contains Actors such as rectangles, images, or text. We will talk more about the actors in the next chapter, but for now let's see how a stage can be created and how we can respond to user interaction with the stage itself.

First make sure that you have called `clutter_init()` to initialize Clutter. You may then get the application's stage with `clutter_stage_get_default()`. This function always returns the same instance, with its own window. You could instead use a `GtkClutterEmbed` widget inside a more complicated GTK+ window - see the Stage Widget section.

`ClutterStage` is derived from the `ClutterActor` object so many of that object's functions are useful for the stage. For instance, call `clutter_actor_show()` to make the stage visible.

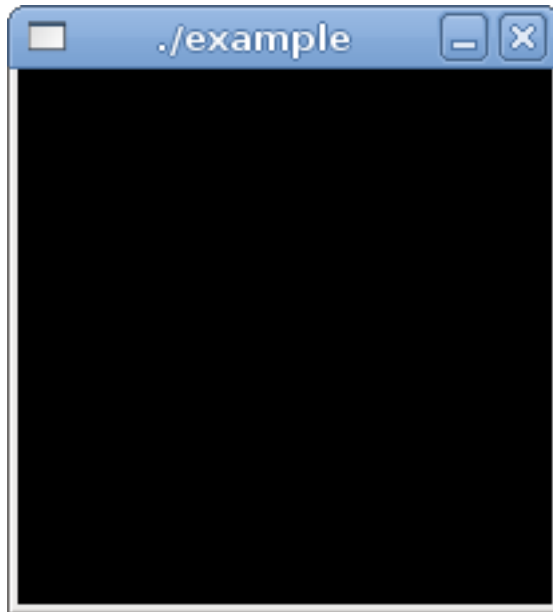
`ClutterStage` also implements the `ClutterContainer` interface, allowing it to contain child actors via calls to `clutter_container_add()`.

Call `clutter_main()` to start a main loop so that the stage can animate its contents and respond to user interaction.

Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterStage.html>)

### 3.1.1. Example

The following example shows a `ClutterStage` and handles clicks on the stage. There are no actors yet so all you will see is a black rectangle.

**Figure 3-1. Stage**

Source Code (../../../../examples/stage)

File: main.c

```
#include <clutter/clutter.h>
#include <stdlib.h>

static gboolean
on_stage_button_press (ClutterStage *stage, ClutterEvent *event, gpointer data)
{
    gint x = 0;
    gint y = 0;
    clutter_event_get_coords (event, &x, &y);

    g_print ("Stage clicked at (%d, %d)\n", x, y);

    return TRUE; /* Stop further handling of this event. */
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff }; /* Black */

    clutter_init (&argc, &argv);
```

```

/* Get the stage and set its size and color: */
ClutterActor *stage = clutter_stage_get_default ();
clutter_actor_set_size (stage, 200, 200);
clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

/* Show the stage: */
clutter_actor_show (stage);

/* Connect a signal handler to handle mouse clicks and key presses on the stage: */
g_signal_connect (stage, "button-press-event",
    G_CALLBACK (on_stage_button_press), NULL);

/* Start the main loop, so we can respond to events: */
clutter_main ();

return EXIT_SUCCESS;
}

```

## 3.2. Stage Widget

The `GtkClutterEmbed` widget allows you to place a `ClutterStage` inside an existing GTK+ window. For instance, the window might contain other GTK+ widgets allowing the user to affect the actors in stage. Use `gtk_clutter_embed_new()` to instantiate the widget and then add it to a container just like any other GTK+ widget. Call `gtk_clutter_embed_get_stage` to get the `ClutterStage` from the `GtkClutterEmbed` widget so you can then use the main Clutter API.

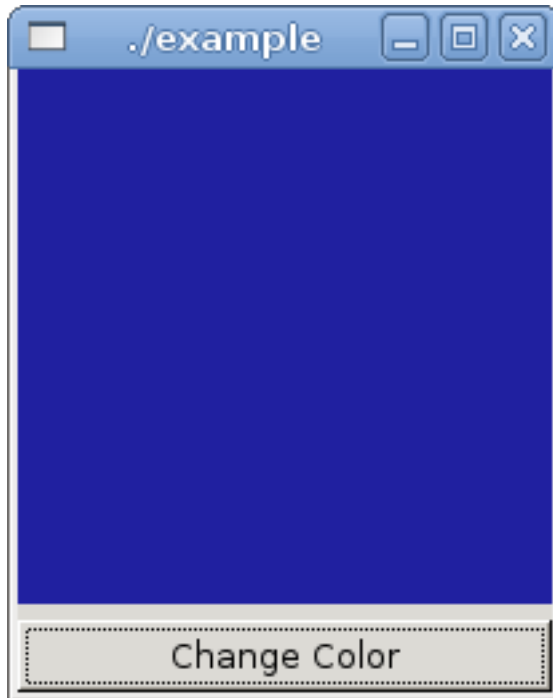
When using the `GtkClutterEmbed` widget you should use the regular `gtk_main()` function to start the mainloop rather than `clutter_main()`.

For simplicity, all other examples in this document will instead use `clutter_stage_get_default()`, but all the techniques can also be used with a stage inside the `GtkClutterEmbed` widget.

Reference (<http://www.clutter-project.org/docs/clutter-ClutterClutter.html>)

### 3.2.1. Example

The following example shows a `GtkClutterEmbed` GTK+ widget and changes the stage color when a button is clicked.

**Figure 3-2. Stage Widget**

Source Code (../../../../examples/stage\_widget)

File: main.c

```
#include <gtk/gtk.h>
#include <clutter/clutter.h>
#include <clutter-gtk/gtk-clutter-embed.h>
#include <stdlib.h>

ClutterActor *stage = NULL;

static gboolean
on_button_clicked (GtkButton *button, gpointer user_data)
{
    static gboolean already_changed = FALSE;
    if(already_changed)
    {
        ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff }; /* Black */
        clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);
    }
    else
    {
```

```

    ClutterColor stage_color = { 0x20, 0x20, 0xA0, 0xff };
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);
}

already_changed = !already_changed;

return TRUE; /* Stop further handling of this event. */
}

static gboolean
on_stage_button_press (ClutterStage *stage, ClutterEvent *event, gpointer user_data)
{
    gint x = 0;
    gint y = 0;
    clutter_event_get_coords (event, &x, &y);

    g_print ("Stage clicked at (%d, %d)\n", x, y);

    return TRUE; /* Stop further handling of this event. */
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff }; /* Black */

    clutter_init (&argc, &argv);
    gtk_init (&argc, &argv);

    /* Create the window and some child widgets: */
    GtkWidget *window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    GtkWidget *vbox = gtk_vbox_new (FALSE, 6);
    gtk_container_add (GTK_CONTAINER (window), vbox);
    gtk_widget_show (vbox);
    GtkWidget *button = gtk_button_new_with_label ("Change Color");
    gtk_box_pack_end (GTK_BOX (vbox), button, FALSE, FALSE, 0);
    gtk_widget_show (button);
    g_signal_connect (button, "clicked",
        G_CALLBACK (on_button_clicked), NULL);

    /* Stop the application when the window is closed: */
    g_signal_connect (window, "hide",
        G_CALLBACK (gtk_main_quit), NULL);

    /* Create the clutter widget: */
    GtkWidget *clutter_widget = gtk_clutter_embed_new ();
    gtk_box_pack_start (GTK_BOX (vbox), clutter_widget, TRUE, TRUE, 0);
    gtk_widget_show (clutter_widget);

    /* Set the size of the widget,
     * because we should not set the size of its stage when using GtkClutterEmbed.
     */
    gtk_widget_set_size_request (clutter_widget, 200, 200);
}

```

```
/* Get the stage and set its size and color: */
stage = gtk_clutter_embed_get_stage (GTK_CLUTTER_EMBED (clutter_widget));
clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

/* Show the stage: */
clutter_actor_show (stage);

/* Connect a signal handler to handle mouse clicks and key presses on the stage: */
g_signal_connect (stage, "button-press-event",
    G_CALLBACK (on_stage_button_press), NULL);

/* Show the window: */
gtk_widget_show (GTK_WIDGET (window));

/* Start the main loop, so we can respond to events: */
gtk_main ();

return EXIT_SUCCESS;
}
```

# Chapter 4. Actors

## 4.1. Actor Basics

As mentioned in the introduction, Clutter is a canvas API for 2D surfaces in 3D space. Standard Clutter actors have 2D shapes and can be positioned and rotated in all three dimensions, but they have no depth. Theoretically, therefore, most actors would be invisible if they were exactly rotated so that only their edge faced the screen. When complex 3D objects are needed, you may use the full OpenGL ES API, as mentioned in the Implementing Actors appendix, but let's look at the standard actors for now:

- `ClutterStage` (<http://clutter-project.org/docs/clutter/0.6/ClutterStage.html>): The stage itself, mentioned already
- `ClutterRectangle` (<http://clutter-project.org/docs/clutter/0.6/ClutterRectangle.html>): A rectangle.
- `ClutterLabel` (<http://clutter-project.org/docs/clutter/0.6/ClutterLabel.html>): Displays text.
- `ClutterEntry` (<http://clutter-project.org/docs/clutter/0.6/ClutterEntry.html>): Text that may be edited by the user.
- `ClutterTexture` (<http://clutter-project.org/docs/clutter/0.6/ClutterTexture.html>): An image.

Each actor should be added to the stage with `clutter_container_add()` and its positions should then be specified. All actors derive from `ClutterActor` so you can call `clutter_actor_set_position()` to set the x and y coordinates, and the z coordinate can be set with `clutter_actor_set_depth()`, with larger values placing the actor further away from the observer. `clutter_actor_set_size()` sets the width and height.

The actor's position is relative to the top-left (0, 0) of the parent container (such as the stage), but this origin can be changed by calling `clutter_actor_set_anchor_point()`.

By default, actors are hidden, so remember to call `clutter_actor_show()`. You may later call `clutter_actor_hide()` to temporarily hide the object again.

Like GTK+ widgets, Clutter actors have a "floating reference" when they are first instantiated with a function such as `clutter_rectangle_new()`. This reference is then taken when the actor is added to a container, such as the stage. This means that you do not need to unreference the actor after creating it.

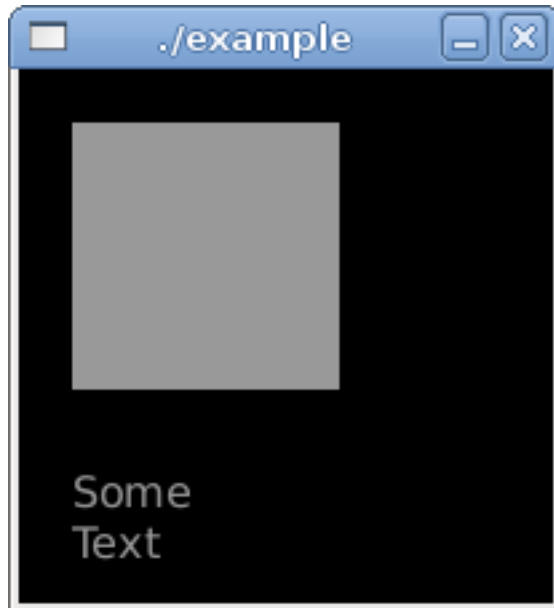
Actors may also be transformed by scaling or rotation, and may be made partly transparent.

Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterActor.html>)

### 4.1.1. Example

The following example demonstrates two unmoving actors in a stage:

**Figure 4-1. Actor**



Source Code ([../../examples/actor](#))

File: main.c

```
#include <clutter/clutter.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor actor_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);
}
```

```

/* Add a rectangle to the stage: */
ClutterActor *rect = clutter_rectangle_new_with_color (&actor_color);
clutter_actor_set_size (rect, 100, 100);
clutter_actor_set_position (rect, 20, 20);
clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
clutter_actor_show (rect);

/* Add a label to the stage: */
ClutterActor *label = clutter_label_new_full ("Sans 12", "Some Text", &actor_color);
clutter_actor_set_size (label, 500, 500);
clutter_actor_set_position (label, 20, 150);
clutter_container_add_actor (CLUTTER_CONTAINER (stage), label);
clutter_actor_show (label);

/* Show the stage: */
clutter_actor_show (stage);

/* Start the main loop, so we can respond to events: */
clutter_main ();

return EXIT_SUCCESS;
}

```

## 4.2. Transformations

Actors can be scaled, rotated, and moved.

### 4.2.1. Scaling

Call `clutter_actor_set_scale()` to increase or decrease the apparent size of the actor. Note that this will not change the result of `clutter_actor_get_width()` and `clutter_actor_get_height()` because it only changes the size of the actor as seen by the user. Calling `clutter_actor_set_scale()` again will replace the first scale rather than multiplying it.

### 4.2.2. Rotation

Call `clutter_actor_set_rotation()` to rotate the actor around an axis, specifying either `CLUTTER_X_AXIS`, `CLUTTER_Y_AXIS` or `CLUTTER_Z_AXIS` and the desired angle. Only two of the x, y, and z coordinates are used, depending on the specified axis. For instance, when using `CLUTTER_X_AXIS`, the y and z parameters specify the center of rotation on the plane of the x axis.

Like the `clutter_actor_set_scale()`, this does not affect the position, width, or height of the actor as returned by functions such as `clutter_actor_get_x()`.

### 4.2.3. Clipping

Actors may be "clipped" so that only one rectangular part of the actor is visible, by calling `clutter_actor_set_clip()`, providing a position relative to the actor, along with the size. For instance, you might implement scrolling by creating a large container actor and setting a clip rectangle so that only a small part of the whole is visible at any one time. Scrolling up could then be implemented by moving the actor down while moving the clip up. Clipping can be reverted by calling `clutter_actor_remove_clip()`.

The area outside of the clip does not consume video memory and generally does not require much processing.

### 4.2.4. Movement

Clutter does not have a translation function that behaves similarly to `clutter_actor_set_scale()` and `clutter_actor_set_rotation()`, but you can move the actor by calling `clutter_actor_move_by()` or `clutter_actor_set_depth()`.

Unlike the scaling and rotation functions, `clutter_actor_move_by()` does change the result of functions such as `clutter_actor_get_x()`.

### 4.2.5. Example

The following example demonstrates two unmoving actors in a stage, using rotation, scaling and movement:

Figure 4-2. Actor



Source Code (`../../../../examples/actor_transformations`)

File: `main.c`

```
#include <clutter/clutter.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor actor_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add a rectangle to the stage: */
    ClutterActor *rect = clutter_rectangle_new_with_color (&actor_color);
    clutter_actor_set_size (rect, 100, 100);
    clutter_actor_set_position (rect, 20, 20);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
}
```

```

clutter_actor_show (rect);

/* Rotate it 20 degrees away from us around the x axis
 * (around its top edge)
 */
clutter_actor_set_rotation (rect, CLUTTER_X_AXIS, -20, 0, 0, 0);

/* Add a label to the stage: */
ClutterActor *label = clutter_label_new_full ("Sans 12", "Some Text", &actor_color);
clutter_actor_set_size (label, 500, 500);
clutter_actor_set_position (label, 20, 150);
clutter_container_add_actor (CLUTTER_CONTAINER (stage), label);
clutter_actor_show (label);

/* Scale it 300% along the x axis:
 */
clutter_actor_set_scale (label, 3.00, 1.0);

/* Move it up and to the right: */
clutter_actor_move_by (label, 10, -10);

/* Move it along the z axis, further from the viewer: */
clutter_actor_set_depth (label, -20);

/* Show the stage: */
clutter_actor_show (stage);

/* Start the main loop, so we can respond to events: */
clutter_main ();

return EXIT_SUCCESS;
}

```

## 4.3. Containers

Some clutter actors implement the `ClutterContainer` interface. These actors can contain child actors and may position them in relation to each other, for instance in a list or a table formation. In addition, transformations or property changes may be applied to all children. Child actors can be added to a container with the `clutter_container_add()` function.

The main `ClutterStage` is itself a container, allowing it to contain all the child actors. The only other container in Core Clutter is `ClutterGroup`, which can contain child actors, with positions relative to the parent `ClutterGroup`. Scaling, rotation and clipping of the group applies to the child actors, which can simplify your code.

Additional Clutter containers can be found in the Tidy toolkit library. See also the Implementing Containers section.

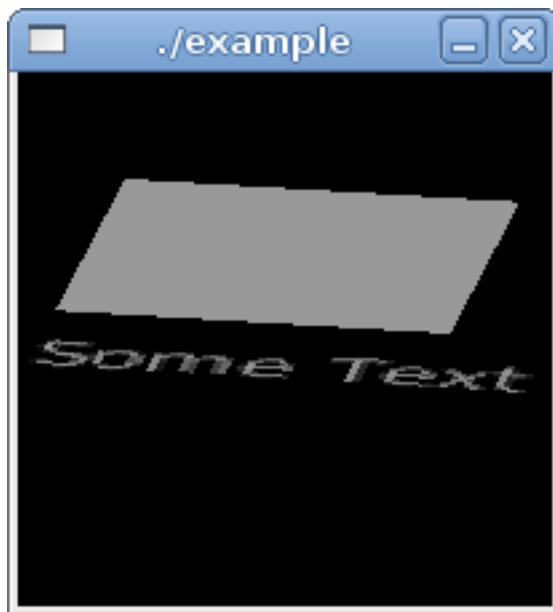
ClutterContainer Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterContainer.html>)

ClutterGroup Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterContainer.html>)

### 4.3.1. Example

The following example shows the use of the `ClutterGroup` container, with two child actors being rotated together.

**Figure 4-3. Group**



Source Code ([../../examples/actor\\_group](#))

File: `main.c`

```
#include <clutter/clutter.h>
#include <stdlib.h>
```

```
int main(int argc, char *argv[])
```

```

{
ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
ClutterColor actor_color = { 0xff, 0xff, 0xff, 0x99 };

clutter_init (&argc, &argv);

/* Get the stage and set its size and color: */
ClutterActor *stage = clutter_stage_get_default ();
clutter_actor_set_size (stage, 200, 200);
clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

/* Add a group to the stage: */
ClutterActor *group = clutter_group_new ();
clutter_actor_set_position (group, 40, 40);
clutter_container_add_actor (CLUTTER_CONTAINER (stage), group);
clutter_actor_show (group);

/* Add a rectangle to the group: */
ClutterActor *rect = clutter_rectangle_new_with_color (&actor_color);
clutter_actor_set_size (rect, 50, 50);
clutter_actor_set_position (rect, 0, 0);
clutter_container_add_actor (CLUTTER_CONTAINER (group), rect);
clutter_actor_show (rect);

/* Add a label to the group: */
ClutterActor *label = clutter_label_new_full ("Sans 9", "Some Text", &actor_color);
clutter_actor_set_position (label, 0, 60);
clutter_container_add_actor (CLUTTER_CONTAINER (group), label);
clutter_actor_show (label);

/* Scale the group 120% along the x axis:
*/
clutter_actor_set_scale (group, 3.00, 1.0);

/* Rotate it around the z axis: */
clutter_actor_set_rotation (group, CLUTTER_Z_AXIS, 10, 0, 0, 0);

/* Show the stage: */
clutter_actor_show (stage);

/* Start the main loop, so we can respond to events: */
clutter_main ();

return EXIT_SUCCESS;
}

```

## 4.4. Events

The base `ClutterActor` has several signals that are emitted when the user interacts with the actor:

- `button-press-event`: Emitted when the user presses the mouse over the actor.
- `button-release-event`: Emitted when the user releases the mouse over the actor.
- `motion-event`: Emitted when the user moves the mouse over the actor.
- `enter-event`: Emitted when the user moves the mouse in to the actor's area.
- `leave-event`: Emitted when the user moves the mouse out of the actor's area.

For instance, you can detect button clicks on an actor like so:

```
g_signal_connect (rect, "button-press-event", G_CALLBACK (on_rect_button_press), NULL);
```

Alternatively, you might just handle signals from the parent `ClutterStage` and use `clutter_stage_get_actor_at_pos` to discover which actor should be affected.

However, as a performance optimization, Clutter does not emit all event signals by default. For instance, to receive event signals for an actor instead of just the stage, you must call

```
clutter_actor_set_reactive(). If you don't need the motion event signals (motion-event, enter-event and leave-event), you may call the global clutter_set_motion_events_enabled() function with FALSE to further optimize performance.
```

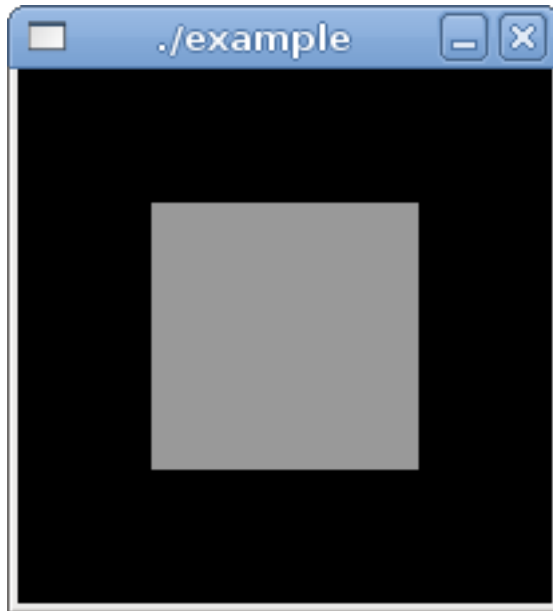
Your event signal handler should return `true` when it has fully handled the event, or `false` if you want the event to be sent also to the next actor in the event chain. Clutter first allows the stage to handle each event via the `captured-event` signal. But if the stage does not handle the event then it will be passed down to the child actor, first passing through the actor's parent containers, giving each actor in the hierarchy a chance to handle the event via a `captured-event` signal handler. If the event has still not been handled fully by any actor then the event will then be emitted via a specific signal (such as `button-press-event` or `key-press-event`). These specific signals are emitted first from the child actor, then by its parent, passing all they way back up to the stage if no signal handler returns `true` to indicate that it has handled the event fully.

Actors usually only receive keyboard events when the actor has key focus, but you can give an actor exclusive access to any events by grabbing either the pointer or the keyboard, using `clutter_grab_pointer` or `clutter_grab_keyboard`.

### 4.4.1. Example

The following example demonstrates handing of clicks on an actor:

**Figure 4-4. Actor Events**



Source Code ([../../examples/actor\\_events](#))

File: main.c

```
#include <clutter/clutter.h>
#include <stdlib.h>

static gboolean
on_stage_button_press (ClutterStage *stage, ClutterEvent *event, gpointer data)
{
    gint x = 0;
    gint y = 0;
    clutter_event_get_coords (event, &x, &y);

    g_print ("Clicked stage at (%d, %d)\n", x, y);

    /* Discover whether there is an actor at that position.
     * Note that you can also connect directly to the actor's signals instead.
     */
    ClutterActor *rect = clutter_stage_get_actor_at_pos (stage, x, y);
```

```

if (!rect)
    return FALSE;

if (CLUTTER_IS_RECTANGLE (rect))
    g_print (" A rectangle is at that position.\n");

return TRUE; /* Stop further handling of this event. */
}

static gboolean
on_rect_button_press (ClutterRectangle *rect, ClutterEvent *event, gpointer data)
{
    gint x = 0;
    gint y = 0;
    clutter_event_get_coords (event, &x, &y);

    g_print ("Clicked rectangle at (%d, %d)\n", x, y);

    /* clutter_main_quit(); */

    return TRUE; /* Stop further handling of this event. */
}

static gboolean
on_rect_button_release (ClutterRectangle *rect, ClutterEvent *event, gpointer data)
{
    gint x = 0;
    gint y = 0;
    clutter_event_get_coords (event, &x, &y);

    g_print ("Click-release on rectangle at (%d, %d)\n", x, y);

    return TRUE; /* Stop further handling of this event. */
}

static gboolean
on_rect_motion (ClutterRectangle *rect, ClutterEvent *event, gpointer data)
{
    g_print ("Motion in the rectangle.\n");

    return TRUE; /* Stop further handling of this event. */
}

static gboolean
on_rect_enter (ClutterRectangle *rect, ClutterEvent *event, gpointer data)
{
    g_print ("Entered rectangle.\n");

    return TRUE; /* Stop further handling of this event. */
}

static gboolean
on_rect_leave (ClutterRectangle *rect, ClutterEvent *event, gpointer data)

```

```

{
    g_print ("Left rectangle.\n");

    return TRUE; /* Stop further handling of this event. */
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor label_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Connect signal handlers to handle mouse clicks and key presses on the stage: */
    g_signal_connect (stage, "button-press-event",
        G_CALLBACK (on_stage_button_press), NULL);

    /* Add a Rectangle to the stage: */
    ClutterActor *rect = clutter_rectangle_new_with_color (&label_color);
    clutter_actor_set_size (rect, 100, 100);
    clutter_actor_set_position (rect, 50, 50);
    clutter_actor_show (rect);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);

    /* Allow the actor to emit events.
     * By default only the stage does this.
     */
    clutter_actor_set_reactive (rect, TRUE);

    /* Connect signal handlers for events: */
    g_signal_connect (rect, "button-press-event",
        G_CALLBACK (on_rect_button_press), NULL);
    g_signal_connect (rect, "button-release-event",
        G_CALLBACK (on_rect_button_release), NULL);
    g_signal_connect (rect, "motion-event",
        G_CALLBACK (on_rect_motion), NULL);
    g_signal_connect (rect, "enter-event",
        G_CALLBACK (on_rect_enter), NULL);
    g_signal_connect (rect, "leave-event",
        G_CALLBACK (on_rect_leave), NULL);

    /* Show the stage: */
    clutter_actor_show (stage);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();
}

```

```
return EXIT_SUCCESS;  
}
```

# Chapter 5. Timelines

## 5.1. Using Timelines

A `ClutterTimeline` can be used to change the position or appearance of an actor over time. These can be used directly as described in this chapter, or together with an effect or behaviour, as you will see in the following chapters.

The timeline object emits its `new-frame` signal for each frame that should be drawn, for as many frames per second as specified. In your signal handler you can set the actor's properties. For instance, the actor might be moved and rotated over time, and its color might change while this is happening. You could even change the properties of several actors to animate the entire stage.

The `clutter_timeline_new()` constructor function takes a number of frames, and a number of frames per second, so the entire timeline will have a duration of  $n\_frames / fps$ . You might therefore choose the number of frames based on a desired duration, by dividing the duration by the desired frames per second.

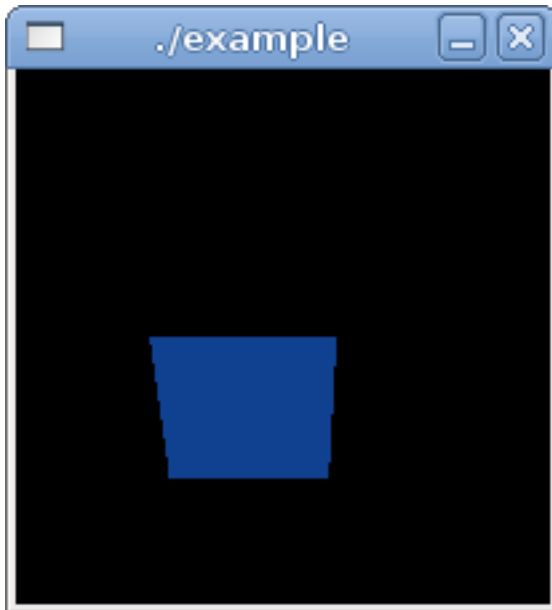
You may also use `clutter_timeline_set_loop()` to cause the timeline to repeat for ever, or until you call `clutter_timeline_stop()`. The timeline does not start until you call `clutter_timeline_start()`.

Remember to unref the timeline when you are finished with it. Unlike actors, this does not have a "floating reference". You may either do this after your mainloop has finished, or when the timeline has finished, by handling the timeline's `completed` signal.

Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterTimeline.html>)

## 5.2. Example

The following example demonstrates the use of a timeline to rotate a rectangle around the x axis while changing its color:

**Figure 5-1. Timeline**

Source Code (../../../../examples/timeline)

File: main.c

```
#include <clutter/clutter.h>
#include <stdlib.h>

ClutterActor *rect = NULL;

gint rotation_angle = 0;
gint color_change_count = 0;

void
on_timeline_new_frame (ClutterTimeline *timeline,
                       gint frame_num, gpointer data)
{
    rotation_angle += 1;
    if(rotation_angle >= 360)
        rotation_angle = 0;

    /* Rotate the rectangle clockwise around the z axis, around it's top-left corner: */
    clutter_actor_set_rotation (rect, CLUTTER_X_AXIS,
                               rotation_angle, 0, 0, 0);

    /* Change the color
```

```

    * (This is a silly example, making the rectangle flash):
    */
    ++color_change_count;
    if(color_change_count > 100)
        color_change_count = 0;

    if(color_change_count == 0)
    {
        ClutterColor rect_color = { 0xff, 0xff, 0xff, 0x99 };
        clutter_rectangle_set_color (CLUTTER_RECTANGLE (rect), &rect_color);
    }
    else if (color_change_count == 50)
    {
        ClutterColor rect_color = { 0x10, 0x40, 0x90, 0xff };
        clutter_rectangle_set_color (CLUTTER_RECTANGLE (rect), &rect_color);
    }
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor rect_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add a rectangle to the stage: */
    rect = clutter_rectangle_new_with_color (&rect_color);
    clutter_actor_set_size (rect, 70, 70);
    clutter_actor_set_position (rect, 50, 100);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
    clutter_actor_show (rect);

    /* Show the stage: */
    clutter_actor_show (stage);

    ClutterTimeline *timeline = clutter_timeline_new(10 /* frames */, 120 /* frames per second */);
    g_signal_connect (timeline, "new-frame", G_CALLBACK (on_timeline_new_frame), NULL);
    clutter_timeline_set_loop(timeline, TRUE);
    clutter_timeline_start(timeline);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();

    g_object_unref (timeline);

    return EXIT_SUCCESS;
}

```

}

## 5.3. Grouping TimeLines in a Score

A `ClutterScore` allows you to start and stop several timelines at once, or run them in sequence one after the other.

To add a timeline that should start first, call `clutter_score_append()` with `NULL` for the parent argument. All such timelines will be started when you call `clutter_score_start()`.

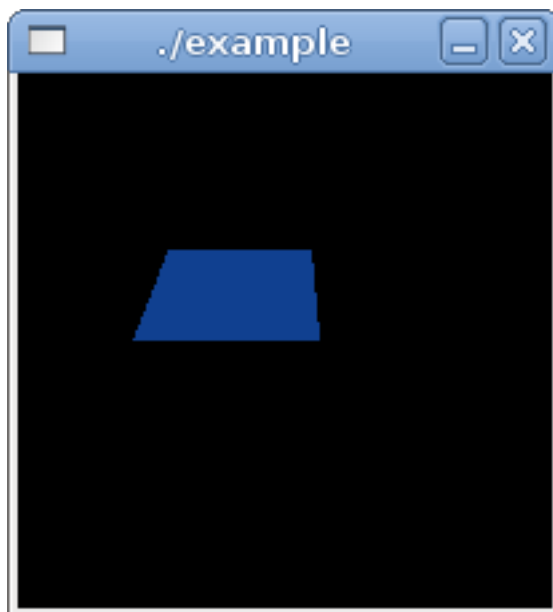
To add a timeline that should be started when a previously added timeline stops, call `clutter_score_append()` with the first timeline for the parent argument.

Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterScore.html>)

## 5.4. Example

The following example demonstrates the use of a score containing two timelines, with one starting when the other ends, and the whole score running as a loop. The first timeline rotates the rectangle as in the previous example, and the second timeline moves the rectangle horizontally.

**Figure 5-2. Score**



## Source Code (../../../../examples/score)

File: main.c

```

#include <clutter/clutter.h>
#include <stdlib.h>

ClutterActor *rect = NULL;

gint rotation_angle = 0;
gint color_change_count = 0;

/* Rotate the rectangle and alternate its color. */
void
on_timeline_rotation_new_frame (ClutterTimeline *timeline,
    gint frame_num, gpointer data)
{
    rotation_angle += 1;
    if(rotation_angle >= 360)
        rotation_angle = 0;

    /* Rotate the rectangle clockwise around the z axis, around it's top-left corner: */
    clutter_actor_set_rotation (rect, CLUTTER_X_AXIS,
        rotation_angle, 0, 0, 0);

    /* Change the color
     * (This is a silly example, making the rectangle flash):
     */
    ++color_change_count;
    if(color_change_count > 100)
        color_change_count = 0;

    if(color_change_count == 0)
    {
        ClutterColor rect_color = { 0xff, 0xff, 0xff, 0x99 };
        clutter_rectangle_set_color (CLUTTER_RECTANGLE (rect), &rect_color);
    }
    else if (color_change_count == 50)
    {
        ClutterColor rect_color = { 0x10, 0x40, 0x90, 0xff };
        clutter_rectangle_set_color (CLUTTER_RECTANGLE (rect), &rect_color);
    }
}

/* Move the rectangle. */
void
on_timeline_move_new_frame (ClutterTimeline *timeline,
    gint frame_num, gpointer data)
{

```

```

gint x_position = clutter_actor_get_x (rect);

x_position += 1;
if(x_position >= 150)
    x_position = 0;

clutter_actor_set_x (rect, x_position);
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor rect_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add a rectangle to the stage: */
    rect = clutter_rectangle_new_with_color (&rect_color);
    clutter_actor_set_size (rect, 70, 70);
    clutter_actor_set_position (rect, 50, 100);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
    clutter_actor_show (rect);

    /* Show the stage: */
    clutter_actor_show (stage);

    /* Create a score and add two timelines to it,
     * so the second timeline starts when the first one stops: */
    ClutterScore *score = clutter_score_new ();
    clutter_score_set_loop (score, TRUE);

    ClutterTimeline *timeline_rotation = clutter_timeline_new (200 /* frames */, 120 /* frame
g_signal_connect (timeline_rotation, "new-frame", G_CALLBACK (on_timeline_rotation_new_fr
clutter_score_append (score, NULL, timeline_rotation);

    ClutterTimeline *timeline_move = clutter_timeline_new (200 /* frames */, 120 /* frames pe
g_signal_connect (timeline_move, "new-frame", G_CALLBACK (on_timeline_move_new_frame), NU
clutter_score_append (score, timeline_rotation, timeline_move);

    clutter_score_start (score);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();

    g_object_unref (timeline_rotation);
    g_object_unref (timeline_move);
    g_object_unref (score);
}

```

```
return EXIT_SUCCESS;  
}
```

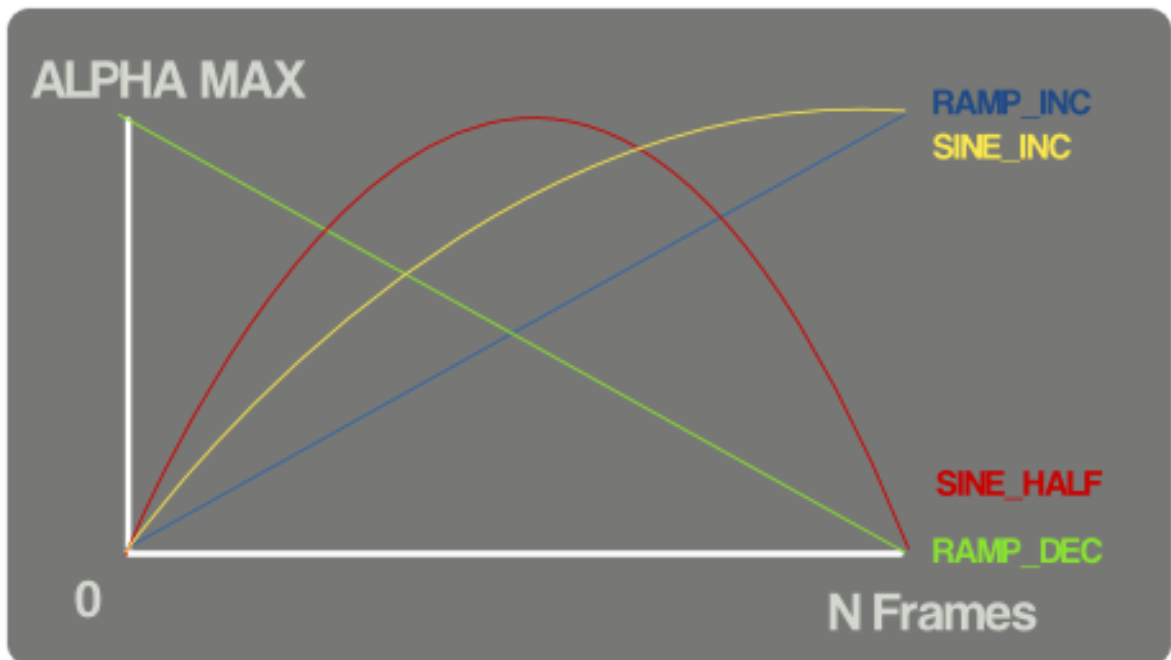
# Chapter 6. Effects

## 6.1. Using Effects

Clutter provides several effects functions that can be used together with a timeline to change the properties of a single actor over time, using a simple numeric calculation. In many cases this is an easier way to implement animation. For instance, `clutter_effect_fade()` gradually changes the opacity of an actor or `clutter_effect_rotate_x()` gradually changes the rotation of an actor, calculating the opacity or rotation by calling the supplied alpha callback.

To use a clutter effect, you should first create a `ClutterEffectTemplate`, specifying your timeline object and a `ClutterAlphaFunc` callback. This callback will need to call `clutter_alpha_get_timeline()` so it can return a value based on the timeline's current frame number and total number of frames, using `clutter_timeline_get_current_frame()` and `clutter_timeline_get_n_frames()`. The result should be between 0 and `CLUTTER_ALPHA_MAX`, with the meaning of the result depending on the effect used. For instance, `CLUTTER_ALPHA_MAX` would be 100% opacity when using `clutter_effect_fade()`. Several built-in callbacks, such as `CLUTTER_ALPHA_SINE`, allow you to easily specify natural movement.

**Figure 6-1. Graphic representation of some alpha functions.**



You should then provide this `ClutterEffectTemplate` to one of the `clutter_effect` functions, along with the actor and any extra parameters required by that function. Remember to unref the effect template after providing it to the effect function (which will take a reference). Unlike actors, these do not have "floating references".

To make it easier to use different timelines with different effects, you can use `clutter_effect_template_set_timeline_clone()` to cause the effect to clone (copy instead of just referencing) the timeline, allowing you to change the original timeline and supply it to a second effect, without influencing the first effect.

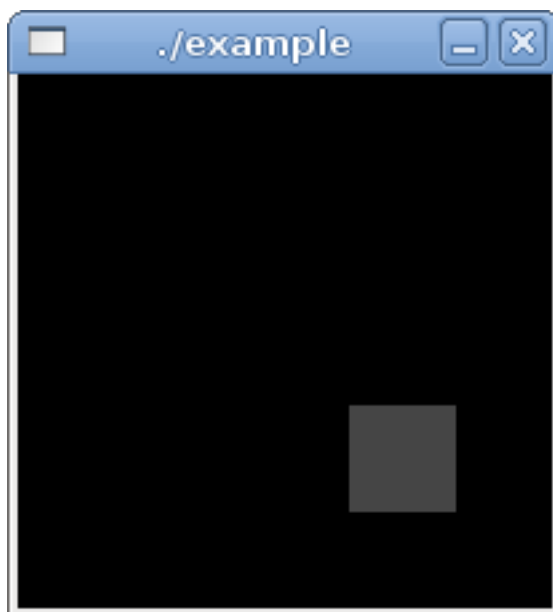
The effects API is actually a simplified API that wraps the `ClutterBehaviour` objects. However, the effect functions can only control one actor at a time and do not allow you to change the effects while the timeline is running. To do this you can use the behaviours directly, as described in the Behaviours section.

Reference (<http://clutter-project.org/docs/clutter/0.6/ClutterClutter-Effects.html>)

## 6.2. Example

The following example demonstrates the use of both a fade effect and a path effect on the same actor, changing a rectangle's opacity while it is moved along a straight line:

**Figure 6-2. Behaviour**



## Source Code (../../examples/effects)

File: main.c

```

#include <clutter/clutter.h>
#include <stdlib.h>

ClutterActor *rect = NULL;

/* This must return a value between 0 and CLUTTER_ALPHA_MAX_ALPHA,
 * where 0 means the start of the path,
 * and CLUTTER_ALPHA_MAX_ALPHA is the end of the path.
 *
 * This will be called as many times per seconds as specified in our call to clutter_timeli
 *
 * See also, for instance CLUTTER_ALPHA_SINE_HALF for a useful built-in callback.
 */
guint32
on_alpha (ClutterAlpha *alpha, gpointer data)
{
    /* Get the position in the timeline,
     * so we can base our value upon it:
     */
    ClutterTimeline *timeline = clutter_alpha_get_timeline (alpha);
    const int current_frame_num = clutter_timeline_get_current_frame (timeline);
    const int n_frames = clutter_timeline_get_n_frames (timeline);

    /* Return a value that is simply proportional to the frame position: */
    return (CLUTTER_ALPHA_MAX_ALPHA * current_frame_num / n_frames);
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor rect_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add a rectangle to the stage: */
    rect = clutter_rectangle_new_with_color (&rect_color);
    clutter_actor_set_size (rect, 40, 40);
    clutter_actor_set_position (rect, 10, 10);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
}

```

```

clutter_actor_show (rect);

/* Show the stage: */
clutter_actor_show (stage);

ClutterTimeline *timeline = clutter_timeline_new(100 /* frames */, 30 /* frames per second */);
clutter_timeline_set_loop(timeline, TRUE);
clutter_timeline_start(timeline);

/* Instead of our custom callback,
 * we could use a standard callback. For instance, CLUTTER_ALPHA_SINE_INC.
 */
ClutterEffectTemplate *effect_template = clutter_effect_template_new (timeline, &on_alpha);

ClutterKnot knot[2];
knot[0].x = 10;
knot[0].y = 10;
knot[1].x = 150;
knot[1].y = 150;

// Move the actor along the path:
clutter_effect_path (effect_template, rect, knot, sizeof(knot) / sizeof(ClutterKnot), NULL);

// Also change the actor's opacity while moving it along the path:
// (You would probably want to use a different ClutterEffectTemplate,
// so you could use a different alpha callback for this.)
clutter_effect_fade (effect_template, rect, 50, NULL, NULL);

g_object_unref (effect_template);
g_object_unref (timeline);

/* Start the main loop, so we can respond to events: */
clutter_main ();

return EXIT_SUCCESS;
}

```

# Chapter 7. Behaviours

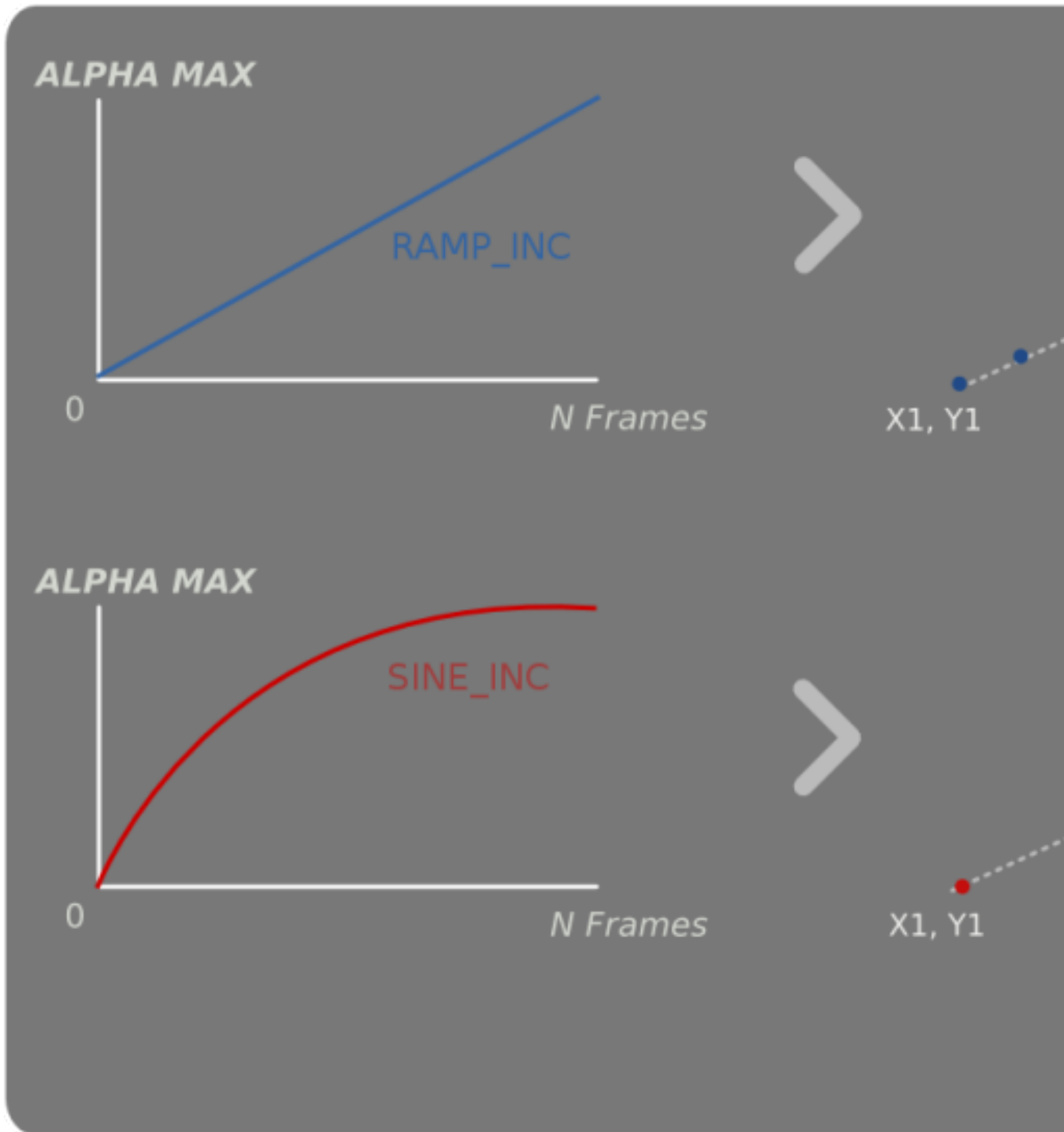
## 7.1. Using Behaviours

The Effects API is simple but you will often need to use behaviours directly to have more control.

Although the `ClutterTimeline`'s `new-frame` signal allows you to set actor properties for each frame, Clutter also provides `Behaviours` which can change specific properties of one specific actor over time, using a simple numeric calculation. However, unlike the simplified Effects API, using behaviours directly allows you to combine them to control multiple actors simultaneously and allows you to change the parameters of the behaviours while the timeline is running.

For instance, `ClutterBehaviourPath` moves the actor along a specified path, calculating the position on the path once per frame by calling a supplied `alpha` callback. The `ClutterAlpha` object is constructed with this callback and a `ClutterTimeline` which tells it when a new frame needs a new value to be calculated.

Figure 7-1. Effects of alpha functions on a path.



Your `alpha` callback will need to call `clutter_alpha_get_timeline()` so it can return a value based on the timeline's current frame number and total number of frames, using `clutter_timeline_get_current_frame` and `clutter_timeline_get_n_frames()`. Several built-in callbacks, such as `CLUTTER_ALPHA_SINE`, allow you to easily specify natural movement.

If the behaviour's timeline is started and not stopped then the end point of the behaviour will always be reached and it will end there unless the timeline is set to loop. For instance, an actor will move along a path until it has reached the end, taking as much time as specified by the timeline's number of frames and frames per second.

Like actors, `ClutterAlpha` has a "floating references" so you don't need to unref it if you have added it to a behaviour. However, the behaviours do not have a floating reference, so you should unref them when you are finished with them. You might do this at the same time as you unref the timeline, for instance in a signal handler for the timeline's `completed` signal.

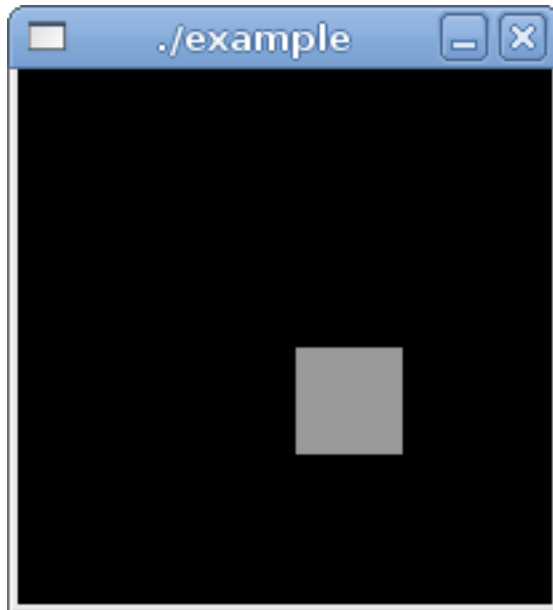
The following standard behaviours are available in Clutter:

- `ClutterBehaviourBspline` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourBspline.html>): Moves the actor along a bezier spline.
- `ClutterBehaviourDepth` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourDepth.html>): Moves the actor along the z axis.
- `ClutterBehaviourEllipse` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourEllipse.html>): Moves the actor around an ellipse.
- `ClutterBehaviourOpacity` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourOpacity.html>): Changes the opacity of the actor.
- `ClutterBehaviourPath` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourPath.html>): Moves the actor along a path defined by a set of points
- `ClutterBehaviourRotate` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourRotate.html>): Rotates the actor.
- `ClutterBehaviourScale` (<http://clutter-project.org/docs/clutter/0.6/ClutterBehaviourScale.html>): Scales the actor.

## 7.2. Example

The following example demonstrates the use of a `ClutterBehaviourPath` with a simple custom `alpha` callback. This simply moves the rectangle from the top-left to the bottom-right of the canvas at constant speed:

Figure 7-2. Behaviour



Source Code (../../examples/behaviour)

File: main.c

```
#include <clutter/clutter.h>
#include <stdlib.h>

ClutterActor *rect = NULL;

/* This must return a value between 0 and CLUTTER_ALPHA_MAX_ALPHA,
 * where 0 means the start of the path,
 * and CLUTTER_ALPHA_MAX_ALPHA is the end of the path.
 *
 * This will be called as many times per seconds as specified in our call to clutter_timeli
 *
 * See also, for instance CLUTTER_ALPHA_SINE_HALF for a useful built-in callback.
 */
guint32
on_alpha (ClutterAlpha *alpha, gpointer data)
{
    /* Get the position in the timeline,
     * so we can base our value upon it:
     */
    ClutterTimeline *timeline = clutter_alpha_get_timeline (alpha);
    const int current_frame_num = clutter_timeline_get_current_frame (timeline);
```

```

const int n_frames = clutter_timeline_get_n_frames (timeline);

/* Return a value that is simply proportional to the frame position: */
return (CLUTTER_ALPHA_MAX_ALPHA * current_frame_num / n_frames);
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor rect_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add a rectangle to the stage: */
    rect = clutter_rectangle_new_with_color (&rect_color);
    clutter_actor_set_size (rect, 40, 40);
    clutter_actor_set_position (rect, 10, 10);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
    clutter_actor_show (rect);

    /* Show the stage: */
    clutter_actor_show (stage);

    ClutterTimeline *timeline = clutter_timeline_new(10 /* frames */, 30 /* frames per second */);
    clutter_timeline_set_loop(timeline, TRUE);
    clutter_timeline_start(timeline);

    /* Instead of our custom callback,
     * we could use a standard callback. For instance, CLUTTER_ALPHA_SINE_INC.
     */
    ClutterAlpha *alpha = clutter_alpha_new_full (timeline, &on_alpha, NULL, NULL);

    ClutterKnot knot[2];
    knot[0].x = 10;
    knot[0].y = 10;
    knot[1].x = 150;
    knot[1].y = 150;

    ClutterBehaviour *behaviour = clutter_behaviour_path_new (alpha, knot, sizeof(knot) / sizeof(knot[0]));
    clutter_behaviour_apply (behaviour, rect);
    g_object_unref (timeline);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();

    return EXIT_SUCCESS;
}

```

}

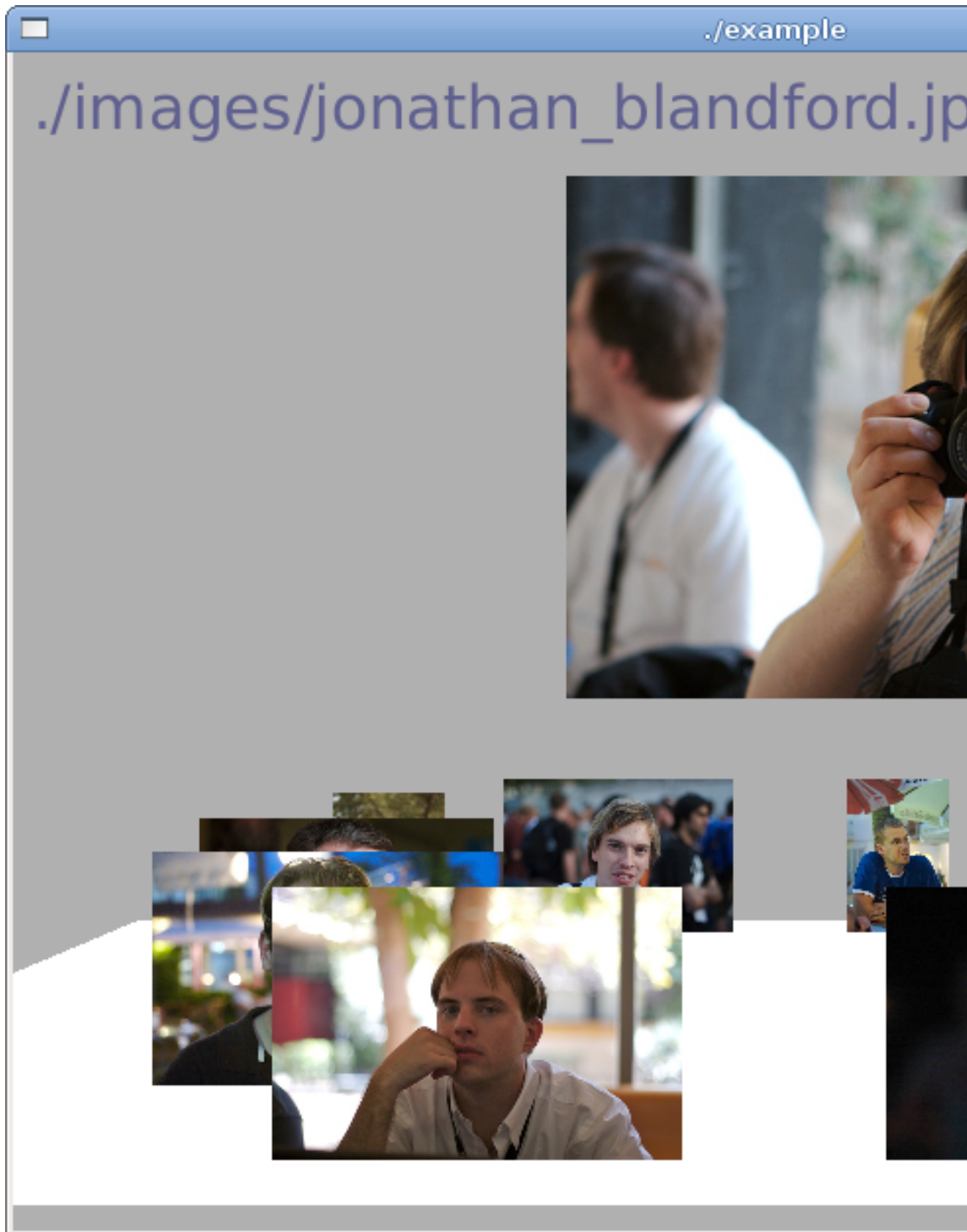
## Chapter 8. Full Example

This example loads images from a directory and displays them in a rotating ellipse. You may click an image to bring it to the front. When the image has rotated to the front it will move up while increasing in size, and the file path will appear at the top of the window.

This is larger than the examples used so far, with multiple timelines and multiple behaviours affecting multiple actors. However, it's still a relatively simple example. A real application would need to be more flexible and have more functionality.

TODO: Make this prettier. Use containers to do that.

Figure 8-1. Full Example



## Source Code (../../examples/full\_example)

**File:** main.c

```

#include <clutter/clutter.h>
#include <stdlib.h>

ClutterActor *stage = NULL;

/* For showing the filename: */
ClutterActor *label_filename = NULL;

/* For rotating all images around an ellipse: */
ClutterTimeline *timeline_rotation = NULL;

/* For moving one image up and scaling it: */
ClutterTimeline *timeline_moveup = NULL;
ClutterBehaviour *behaviour_scale = NULL;
ClutterBehaviour *behaviour_path = NULL;
ClutterBehaviour *behaviour_opacity = NULL;

/* The y position of the ellipse of images: */
const gint ELLIPSE_Y = 390;
const gint ELLIPSE_HEIGHT = 450; /* The distance from front to back when it's rotated 90 de
const gint IMAGE_HEIGHT = 100;

static gboolean
on_texture_button_press (ClutterActor *actor, ClutterEvent *event, gpointer data);

const double angle_step = 30;

typedef struct Item
{
    ClutterActor *actor;
    ClutterBehaviour *ellipse_behaviour;
    gchar* filepath;
}
Item;

Item* item_at_front = NULL;

GSLList *list_items = 0;

void on_foreach_clear_list_items(gpointer data, gpointer user_data)
{
    Item* item = (Item*)data;

    /* We don't need to unref the actor because the floating reference was taken by the stage
    g_object_unref (item->ellipse_behaviour);

```

```

    g_free (item->filepath);
    g_free (item);
}

void scale_texture_default(ClutterActor *texture)
{
    GdkPixbuf *pixbuf = clutter_texture_get_pixbuf (CLUTTER_TEXTURE (texture));
    g_return_if_fail (pixbuf);

    const int pixbuf_height = gdk_pixbuf_get_height(pixbuf);
    const gdouble scale = pixbuf_height ? IMAGE_HEIGHT / (gdouble)pixbuf_height : 0;
    clutter_actor_set_scale (texture, scale, scale);
}

void load_images(const gchar* directory_path)
{
    g_return_if_fail(directory_path);

    /* Clear any existing images: */
    g_slist_foreach (list_items, on_foreach_clear_list_items, NULL);
    g_slist_free (list_items);

    /* Create a new list: */
    list_items = NULL;

    /* Discover the images in the directory: */
    GError *error = NULL;
    GDir* dir = g_dir_open (directory_path, 0, &error);
    if (error)
    {
        g_warning("g_dir_open() failed: %s\n", error->message);
        g_clear_error(&error);
        return;
    }

    const gchar* filename = NULL;
    while ( (filename = g_dir_read_name(dir)) )
    {
        gchar* path = g_build_filename (directory_path, filename, NULL);

        /* Try to load the file as an image: */
        GdkPixbuf* pixbuf = gdk_pixbuf_new_from_file(path, NULL);
        if(pixbuf)
        {
            Item* item = g_new0(Item, 1);

            /* Add an actor to show this image: */
            item->actor = clutter_texture_new_from_pixbuf (pixbuf);
            item->filepath = g_strdup(path);

            /* Make sure that all images are shown with the same height: */
            scale_texture_default (item->actor);
        }
    }
}

```

```

    list_items = g_slist_append (list_items, item);
}

if(pixbuf)
    g_object_unref (pixbuf);

g_free (path);
}
}

void add_to_ellipse_behaviour(ClutterTimeline *timeline_rotation, gdouble start_angle, Item
{
    g_return_if_fail (timeline_rotation);

    ClutterAlpha *alpha = clutter_alpha_new_full (timeline_rotation, CLUTTER_ALPHA_SINE_INC,

item->ellipse_behaviour = clutter_behaviour_ellipse_new (alpha,
    320, ELLIPSE_Y, /* x, y */
    ELLIPSE_HEIGHT, ELLIPSE_HEIGHT, /* width, height */
    CLUTTER_ROTATE_CW,
    start_angle, start_angle + 360);
    clutter_behaviour_ellipse_set_angle_tilt (CLUTTER_BEHAVIOUR_ELLIPSE (item->ellipse_behavi
    CLUTTER_X_AXIS, -90);
    /* Note that ClutterAlpha has a floating reference, so we don't need to unref it. */

    clutter_behaviour_apply (item->ellipse_behaviour, item->actor);
}

void add_image_actors()
{
    int x = 20;
    int y = 0;
    gdouble angle = 0;
    GSList *list = list_items;
    while (list)
    {
        /* Add the actor to the stage: */
        Item *item = (Item*)list->data;
        ClutterActor *actor = item->actor;
        clutter_container_add_actor (CLUTTER_CONTAINER (stage), actor);

        /* Set an initial position: */
        clutter_actor_set_position (actor, x, y);
        y += 100;

        /* Allow the actor to emit events.
        * By default only the stage does this.
        */
        clutter_actor_set_reactive (actor, TRUE);

        /* Connect signal handlers for events: */
        g_signal_connect (actor, "button-press-event",

```

```

        G_CALLBACK (on_texture_button_press), item);

    add_to_ellipse_behaviour (timeline_rotation, angle, item);
    angle += angle_step;

    clutter_actor_show (actor);

    list = g_slist_next (list);
}
}

gdouble angle_in_360(gdouble angle)
{
    gdouble result = angle;
    while(result >= 360)
        result -= 360;

    return result;
}

/* This signal handler is called when the item has finished
 * moving up and increasing in size.
 */
void on_timeline_moveup_completed(ClutterTimeline* timeline, gpointer user_data)
{
    /* Unref this timeline because we have now finished with it: */
    g_object_unref (timeline_moveup);
    timeline_moveup = NULL;

    g_object_unref (behaviour_scale);
    behaviour_scale = NULL;

    g_object_unref (behaviour_path);
    behaviour_path = NULL;

    g_object_unref (behaviour_opacity);
    behaviour_opacity = NULL;
}

/* This signal handler is called when the items have completely
 * rotated around the ellipse.
 */
void on_timeline_rotation_completed(ClutterTimeline* timeline, gpointer user_data)
{
    /* All the items have now been rotated so that the clicked item is at the front. */
    /* Now we transform just this one item gradually some more,
     * and show the filename: */

    /* Transform the image: */
    ClutterActor *actor = item_at_front->actor;
    timeline_moveup = clutter_timeline_new(60 /* frames */, 30 /* frames per second. */);
    ClutterAlpha *alpha = clutter_alpha_new_full (timeline_moveup, CLUTTER_ALPHA_SINE_INC, NU

```

```

/* Scale the item from its normal scale to approximately twice the normal scale: */
gdouble scale_start = 0;
clutter_actor_get_scale (actor, &scale_start, NULL);
const gdouble scale_end = scale_start * 1.8;

behaviour_scale =
    clutter_behaviour_scale_new (alpha, scale_start, scale_start, scale_end, scale_end);
clutter_behaviour_apply (behaviour_scale, actor);

/* Move the item up the y axis: */
ClutterKnot knots[2];
knots[0].x = clutter_actor_get_x (actor);
knots[0].y = clutter_actor_get_y (actor);
knots[1].x = knots[0].x;
knots[1].y = knots[0].y - 250;
behaviour_path =
    clutter_behaviour_path_new (alpha, knots, G_N_ELEMENTS(knots));
clutter_behaviour_apply (behaviour_path, actor);

/* Show the filename gradually: */
clutter_label_set_text (CLUTTER_LABEL (label_filename), item_at_front->filepath);
behaviour_opacity =
    clutter_behaviour_opacity_new (alpha, 0, 255);
clutter_behaviour_apply (behaviour_opacity, label_filename);

/* Start the timeline and handle its "completed" signal so we can unref it. */
g_signal_connect (timeline_moveup, "completed", G_CALLBACK (on_timeline_moveup_completed)
clutter_timeline_start (timeline_moveup);

/* Note that ClutterAlpha has a floating reference so we don't need to unref it. */
}

void rotate_all_until_item_is_at_front (Item *item)
{
    g_return_if_fail (item);

    clutter_timeline_stop(timeline_rotation);

    /* Stop the other timeline in case that is active at the same time: */
    if(timeline_moveup)
        clutter_timeline_stop (timeline_moveup);
    clutter_actor_set_opacity (label_filename, 0);

    /* Get the item's position in the list: */
    const gint pos = g_slist_index (list_items, item);
    g_assert (pos != -1);

    if(!item_at_front && list_items)
        item_at_front = (Item*)list_items->data;
}

```

```

gint pos_front = 0;
if(item_at_front)
    pos_front = g_slist_index (list_items, item_at_front);
g_assert (pos_front != -1);

/* const gint pos_offset_before_start = pos_front - pos; */

/* Calculate the end angle of the first item: */
const gdouble angle_front = 180;
gdouble angle_start = angle_front - (angle_step * pos_front);
angle_start = angle_in_360 (angle_start);
gdouble angle_end = angle_front - (angle_step * pos);

gdouble angle_diff = 0;

/* Set the end angles: */
GSList *list = list_items;
while (list)
{
    Item *this_item = (Item*)list->data;

    /* Reset its size: */
    scale_texture_default (this_item->actor);

    angle_start = angle_in_360 (angle_start);
    angle_end = angle_in_360 (angle_end);

    /* Move 360 instead of 0
     * when moving for the first time,
     * and when clicking on something that is already at the front.
     */
    if(item_at_front == item)
        angle_end += 360;

    clutter_behaviour_ellipse_set_angle_start (
        CLUTTER_BEHAVIOUR_ELLIPSE (this_item->ellipse_behaviour), angle_start);
    clutter_behaviour_ellipse_set_angle_end (
        CLUTTER_BEHAVIOUR_ELLIPSE (this_item->ellipse_behaviour), angle_end);

    if(this_item == item)
    {
        if(angle_start < angle_end)
            angle_diff = angle_end - angle_start;
        else
            angle_diff = 360 - (angle_start - angle_end);

        /* printf(" debug: angle diff=%f\n", angle_diff); */
    }

    /* TODO: Set the number of frames, depending on the angle.
     * otherwise the actor will take the same amount of time to reach
     * the end angle regardless of how far it must move, causing it to

```

```

    * move very slowly if it does not have far to move.
    */
    angle_end += angle_step;
    angle_start += angle_step;
    list = g_slist_next (list);
}

/* Set the number of frames to be proportional to the distance to travel,
   so the speed is always the same: */
gint pos_to_move = 0;
if(pos_front < pos)
{
    const gint count = g_slist_length (list_items);
    pos_to_move = count + (pos - pos_front);
}
else
{
    pos_to_move = pos_front - pos;
}

clutter_timeline_set_n_frames (timeline_rotation, angle_diff);

/* Remember what item will be at the front when this timeline finishes: */
item_at_front = item;

clutter_timeline_start (timeline_rotation);
}

static gboolean
on_texture_button_press (ClutterActor *actor, ClutterEvent *event, gpointer user_data)
{
    /* Ignore the events if the timeline_rotation is running (meaning, if the objects are moving)
       * to simplify things:
       */
    if(timeline_rotation && clutter_timeline_is_playing (timeline_rotation))
    {
        printf("on_texture_button_press(): ignoring\n");
        return FALSE;
    }
    else
        printf("on_texture_button_press(): handling\n");

    Item *item = (Item*)user_data;
    rotate_all_until_item_is_at_front (item);

    return TRUE;
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0xB0, 0xB0, 0xB0, 0xff }; /* light gray */

    clutter_init (&argc, &argv);

```

```

/* Get the stage and set its size and color: */
stage = clutter_stage_get_default ();
clutter_actor_set_size (stage, 800, 600);
clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

/* Create and add a label actor, hidden at first: */
label_filename = clutter_label_new ();
ClutterColor label_color = { 0x60, 0x60, 0x90, 0xff }; /* blueish */
clutter_label_set_color (CLUTTER_LABEL (label_filename), &label_color);
clutter_label_set_font_name (CLUTTER_LABEL (label_filename), "Sans 24");
clutter_actor_set_position (label_filename, 10, 10);
clutter_actor_set_opacity (label_filename, 0);
clutter_container_add_actor (CLUTTER_CONTAINER (stage), label_filename);
clutter_actor_show (label_filename);

/* Add a plane under the ellipse of images: */
ClutterColor rect_color = { 0xff, 0xff, 0xff, 0xff }; /* white */
ClutterActor *rect = clutter_rectangle_new_with_color (&rect_color);
clutter_actor_set_height (rect, ELLIPSE_HEIGHT + 20);
clutter_actor_set_width (rect, clutter_actor_get_width (stage) + 100);
/* Position it so that its center is under the images: */
clutter_actor_set_position (rect,
    -(clutter_actor_get_width (rect) - clutter_actor_get_width (stage)) / 2,
    ELLIPSE_Y + IMAGE_HEIGHT - (clutter_actor_get_height (rect) / 2));
/* Rotate it around its center: */
clutter_actor_set_rotation (rect, CLUTTER_X_AXIS, -90, 0, (clutter_actor_get_height (rect) / 2));
clutter_container_add_actor (CLUTTER_CONTAINER (stage), rect);
clutter_actor_show (rect);

/* Show the stage: */
clutter_actor_show (stage);

timeline_rotation = clutter_timeline_new(60 /* frames */, 30 /* frames per second. */);
g_signal_connect (timeline_rotation, "completed", G_CALLBACK (on_timeline_rotation_completed));

/* Add an actor for each image: */
load_images ("./images/");
add_image_actors ();

/* clutter_timeline_set_loop(timeline_rotation, TRUE); */

/* Move them a bit to start with: */
if(list_items)
    rotate_all_until_item_is_at_front ((Item*)list_items->data);

/* Start the main loop, so we can respond to events: */
clutter_main ();

/* Free the list items and the list: */
g_slist_foreach(list_items, on_foreach_clear_list_items, NULL);

```

```
g_slist_free (list_items);  
  
g_object_unref (timeline_rotation);  
  
return EXIT_SUCCESS;  
  
}
```

# Appendix A. Implementing Actors

## A.1. Implementing Simple Actors

If the standard Clutter actors don't meet all your needs then you may create your own custom actor objects. Implementing a custom actor is much like implementing any new GObject type. You may use the `G_DEFINE_TYPE` macro to specify that the type is derived from `ClutterActor`. For instance:

```
G_DEFINE_TYPE (ClutterTriangle, clutter_triangle, CLUTTER_TYPE_ACTOR);
```

You should then specify your object's implementation of the `ClutterActor::paint()` virtual function in your `class_init` function:

```
static void
clutter_triangle_class_init (ClutterTriangleClass *klass)
{
    ClutterActorClass *actor_class = CLUTTER_ACTOR_CLASS (klass);

    actor_class->paint = clutter_triangle_paint;

    ...
}
```

Your `ClutterActor::paint()` implementation should use the OpenGL API to actually paint something. You will probably need some information from your object's generic `ClutterActor` base class, for instance by calling `clutter_actor_get_geometry()` and `clutter_actor_get_opacity()`, and by using your object's specific property values.

To make your code work with both OpenGL ES and regular OpenGL (and maybe even future Clutter backends), you may wish to use Clutter's `cogl` abstraction API which provides functions such as `cogl_rectangle()` and `cogl_push_matrix()`. You can also detect whether the platform has support for either the OpenGL or OpenGL ES API by `ifdefing` for `CLUTTER_COGL_HAS_GL` or `CLUTTER_COGL_HAS_GLES`.

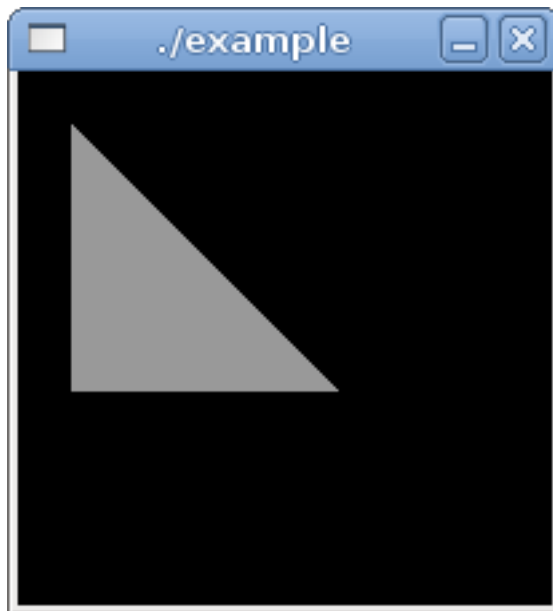
You should also implement the `ClutterActor::pick()` virtual function, painting a silhouette of your actor in the provided color. Clutter uses this to draw each actor's silhouette offscreen in a unique color, using the color to quickly identify the actor under the cursor. If your actor is simple then you can probably reuse the code from your `paint()` implementation.

Most of the rest of `ClutterActor`'s virtual functions don't need to be reimplemented, because a suitable default implementation exists in `ClutterActor`. For instance, `show()`, `show_all()`, `hide()`, `hide_all()`, `request_coord()`.

## A.2. Example

The following example demonstrates the implementation of a new triangle Actor type.

**Figure A-1. Behaviour**



Source Code (`../../examples/custom_actor`)

File: `triangle_actor.h`

```
#ifndef _CLUTTER_TUTORIAL_TRIANGLE_ACTOR_H
#define _CLUTTER_TUTORIAL_TRIANGLE_ACTOR_H

#include <glib-object.h>
#include <clutter/clutter-actor.h>
#include <clutter/clutter-color.h>

G_BEGIN_DECLS

#define CLUTTER_TYPE_TRIANGLE clutter_triangle_get_type()
```

```

#define CLUTTER_TRIANGLE(obj) \
    (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
    CLUTTER_TYPE_TRIANGLE, ClutterTriangle))

#define CLUTTER_TRIANGLE_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_CAST ((klass), \
    CLUTTER_TYPE_TRIANGLE, ClutterTriangleClass))

#define CLUTTER_IS_TRIANGLE(obj) \
    (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
    CLUTTER_TYPE_TRIANGLE))

#define CLUTTER_IS_TRIANGLE_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_TYPE ((klass), \
    CLUTTER_TYPE_TRIANGLE))

#define CLUTTER_TRIANGLE_GET_CLASS(obj) \
    (G_TYPE_INSTANCE_GET_CLASS ((obj), \
    CLUTTER_TYPE_TRIANGLE, ClutterTriangleClass))

typedef struct _ClutterTriangle      ClutterTriangle;
typedef struct _ClutterTriangleClass ClutterTriangleClass;
typedef struct _ClutterTrianglePrivate ClutterTrianglePrivate;

struct _ClutterTriangle
{
    ClutterActor      parent;

    /*< private >*/
    ClutterTrianglePrivate *priv;
};

struct _ClutterTriangleClass
{
    ClutterActorClass parent_class;
};

GType clutter_triangle_get_type (void) G_GNUC_CONST;

ClutterActor *clutter_triangle_new          (void);
ClutterActor *clutter_triangle_new_with_color (const ClutterColor *color);

void          clutter_triangle_get_color    (ClutterTriangle *triangle,
                                             ClutterColor      *color);
void          clutter_triangle_set_color    (ClutterTriangle *triangle,
                                             const ClutterColor *color);

G_END_DECLS

#endif

```

**File:** triangle\_actor.c

```

#include "triangle_actor.h"

#include "clutter/cogl.h" /* For some helper functions. */
#include <GL/glx.h> /* For the OpenGL API. */

G_DEFINE_TYPE (ClutterTriangle, clutter_triangle, CLUTTER_TYPE_ACTOR);

enum
{
    PROP_0,

    PROP_COLOR
};

#define CLUTTER_TRIANGLE_GET_PRIVATE(obj) \
(G_TYPE_INSTANCE_GET_PRIVATE ((obj), CLUTTER_TYPE_TRIANGLE, ClutterTrianglePrivate))

struct _ClutterTrianglePrivate
{
    ClutterColor color;
};

static void
do_triangle_paint (ClutterActor *self, const ClutterColor *color)
{
    ClutterTriangle      *triangle = CLUTTER_TRIANGLE(self);
    ClutterTrianglePrivate *priv;
    ClutterGeometry      geom;

    triangle = CLUTTER_TRIANGLE(self);
    priv = triangle->priv;

    cogl_push_matrix();

    cogl_enable (CGL_ENABLE_BLEND);

    clutter_actor_get_geometry (self, &geom);

    cogl_color (color);

    /* Paint a triangle:
     *
     * The parent paint call will have translated us into position so
     * paint from 0, 0
     *
     * Note that we should really check that glGetError() != GL_NO_ERROR
     * after each gl call, but that would complicate this example. */
    glBegin(GL_POLYGON);
    glVertex2i(0, 0);
    glVertex2i(0, geom.height);
    glVertex2i(geom.width, geom.height);

```

```

    glEnd();

    cogl_pop_matrix();
}

static void
clutter_triangle_paint (ClutterActor *self)
{
    ClutterTriangle *triangle = CLUTTER_TRIANGLE(self);
    ClutterTrianglePrivate *priv = triangle->priv;

    /* Paint the triangle with the actor's color: */
    ClutterColor color;
    color.red   = priv->color.red;
    color.green = priv->color.green;
    color.blue  = priv->color.blue;
    color.alpha = clutter_actor_get_opacity (self);

    do_triangle_paint (self, &color);
}

static void
clutter_triangle_pick (ClutterActor *self, const ClutterColor *color)
{
    /* Paint the triangle with the pick color, offscreen.
       This is used by Clutter to detect the actor under the cursor
       by identifying the unique color under the cursor. */
    do_triangle_paint (self, color);
}

static void
clutter_triangle_set_property (GObject      *object,
                               guint        prop_id,
                               const GValue *value,
                               GParamSpec   *pspec)
{
    ClutterTriangle *triangle = CLUTTER_TRIANGLE(object);

    switch (prop_id)
    {
        case PROP_COLOR:
            clutter_triangle_set_color (triangle, g_value_get_boxed (value));
            break;
        default:
            G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
            break;
    }
}

static void
clutter_triangle_get_property (GObject      *object,
                               guint        prop_id,
                               GValue       *value,

```

```

GParamSpec *pspec)
{
    ClutterTriangle *triangle = CLUTTER_TRIANGLE(object);
    ClutterColor      color;

    switch (prop_id)
    {
        case PROP_COLOR:
            clutter_triangle_get_color (triangle, &color);
            g_value_set_boxed (value, &color);
            break;
        default:
            G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
            break;
    }
}

static void
clutter_triangle_finalize (GObject *object)
{
    G_OBJECT_CLASS (clutter_triangle_parent_class)->finalize (object);
}

static void
clutter_triangle_dispose (GObject *object)
{
    G_OBJECT_CLASS (clutter_triangle_parent_class)->dispose (object);
}

static void
clutter_triangle_class_init (ClutterTriangleClass *klass)
{
    GObjectClass      *gobject_class = G_OBJECT_CLASS (klass);
    ClutterActorClass *actor_class = CLUTTER_ACTOR_CLASS (klass);

    /* Provide implementations for ClutterActor vfuncs: */
    actor_class->paint = clutter_triangle_paint;
    actor_class->pick = clutter_triangle_pick;

    gobject_class->finalize      = clutter_triangle_finalize;
    gobject_class->dispose       = clutter_triangle_dispose;
    gobject_class->set_property  = clutter_triangle_set_property;
    gobject_class->get_property  = clutter_triangle_get_property;

    /**
     * ClutterTriangle:color:
     *
     * The color of the triangle.
     */
    g_object_class_install_property (gobject_class,
                                    PROP_COLOR,

```

```

        g_param_spec_boxed ("color",
                           "Color",
                           "The color of the triangle",
                           CLUTTER_TYPE_COLOR,
                           G_PARAM_READABLE | G_PARAM_WRITABLE)

    g_type_class_add_private (gobject_class, sizeof (ClutterTrianglePrivate));
}

static void
clutter_triangle_init (ClutterTriangle *self)
{
    ClutterTrianglePrivate *priv;

    self->priv = priv = CLUTTER_TRIANGLE_GET_PRIVATE (self);

    priv->color.red = 0xff;
    priv->color.green = 0xff;
    priv->color.blue = 0xff;
    priv->color.alpha = 0xff;
}

/**
 * clutter_triangle_new:
 *
 * Creates a new #ClutterActor with a rectangular shape.
 *
 * Return value: a new #ClutterActor
 */
ClutterActor*
clutter_triangle_new (void)
{
    return g_object_new (CLUTTER_TYPE_TRIANGLE, NULL);
}

/**
 * clutter_triangle_new_with_color:
 * @color: a #ClutterColor
 *
 * Creates a new #ClutterActor with a rectangular shape
 * and with @color.
 *
 * Return value: a new #ClutterActor
 */
ClutterActor *
clutter_triangle_new_with_color (const ClutterColor *color)
{
    return g_object_new (CLUTTER_TYPE_TRIANGLE,
                        "color", color,
                        NULL);
}

/**

```

```

* clutter_triangle_get_color:
* @triangle: a #ClutterTriangle
* @color: return location for a #ClutterColor
*
* Retrieves the color of @triangle.
*/
void
clutter_triangle_get_color (ClutterTriangle *triangle,
                           ClutterColor   *color)
{
    ClutterTrianglePrivate *priv;

    g_return_if_fail (CLUTTER_IS_TRIANGLE (triangle));
    g_return_if_fail (color != NULL);

    priv = triangle->priv;

    color->red = priv->color.red;
    color->green = priv->color.green;
    color->blue = priv->color.blue;
    color->alpha = priv->color.alpha;
}

/**
* clutter_triangle_set_color:
* @triangle: a #ClutterTriangle
* @color: a #ClutterColor
*
* Sets the color of @triangle.
*/
void
clutter_triangle_set_color (ClutterTriangle *triangle,
                           const ClutterColor *color)
{
    ClutterTrianglePrivate *priv;

    g_return_if_fail (CLUTTER_IS_TRIANGLE (triangle));
    g_return_if_fail (color != NULL);

    g_object_ref (triangle);

    priv = triangle->priv;

    priv->color.red = color->red;
    priv->color.green = color->green;
    priv->color.blue = color->blue;
    priv->color.alpha = color->alpha;

    clutter_actor_set_opacity (CLUTTER_ACTOR (triangle),
                              priv->color.alpha);

    if (CLUTTER_ACTOR_IS_VISIBLE (CLUTTER_ACTOR (triangle)))
        clutter_actor_queue_redraw (CLUTTER_ACTOR (triangle));
}

```

```

g_object_notify (G_OBJECT (triangle), "color");
g_object_unref (triangle);
}

```

**File:** main.c

```

#include <clutter/clutter.h>
#include "triangle_actor.h"
#include <stdlib.h>

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor actor_color = { 0xff, 0xff, 0xff, 0x99 };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add our custom actor to the stage: */
    ClutterActor *actor = clutter_triangle_new_with_color (&actor_color);
    clutter_actor_set_size (actor, 100, 100);
    clutter_actor_set_position (actor, 20, 20);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), actor);
    clutter_actor_show (actor);

    /* Show the stage: */
    clutter_actor_show (stage);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();

    return EXIT_SUCCESS;
}

```

## A.3. Implementing Container Actors

You can create container actors that arrange child actors by implementing the `ClutterContainer` interface in your actor. You may use the `G_DEFINE_TYPE_WITH_CODE` macro to specify that the type is derived from `ClutterActor` and also implements the `ClutterContainer` interface. For instance:

```
static void clutter_container_iface_init (ClutterContainerIface *iface);

G_DEFINE_TYPE_WITH_CODE (ExampleContainer, example_container, CLUTTER_TYPE_ACTOR,
                        G_IMPLEMENT_INTERFACE (CLUTTER_TYPE_CONTAINER,
                                              clutter_container_iface_init));
```

### A.3.1. ClutterActor virtual functions to implement

If your container should control the position or size of its children then it should implement the `ClutterActor`'s `request_coords()` and `query_coords()` virtual functions.

For instance, your `request_coords()` implementation should store the provided allocation and maybe use it to layout its child actors, by calling `clutter_actor_request_coords()` on them with appropriate allocations.

Your `query_coords()` implementation should return the position and size desired by your container, by setting the coordinates in the `ClutterActorBox` output parameter. Depending on your container, this might be dependent on the child actors. For instance, your container might increase in size when it contains more child actors, ignoring any calls to `clutter_actor_set_size()`. Alternatively, your container might change the size of its child actors to fit the specified size.

You should implement the `paint()` function, usually calling `clutter_actor_paint()` on the child actors. All containers should also implement the `pick()` function, calling `clutter_actor_pick()` on each child actor.

See the Custom Actor section for more details these virtual functions.

### A.3.2. ClutterContainer virtual functions to implement

Your container implementation should also implement some of the `ClutterContainer` virtual functions so that the container's children will be affected appropriately when functions are called on the container.

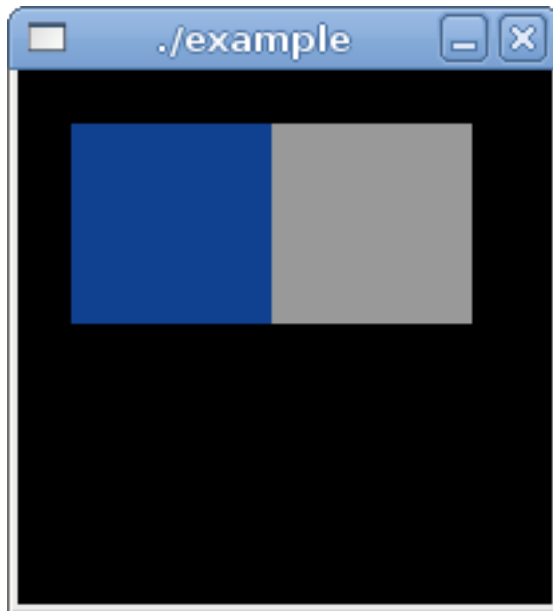
For instance, your `add()` and `remove()` implementations should manage your container's internal list of child actors and might need to trigger repositioning or resizing of the child actors. This repositioning would usually reuse the same algorithm that you used in your `request_coords()` implementation.

Your `foreach` implementation would simply call the provided callback on your container's list of child actors. Note that your container could actually contain additional actors that are not considered to be child actors for the purposes of `add()`, `remove()`, and `foreach()`.

## A.4. Example

The following example demonstrates the implementation of a box container that lays its child actors out horizontally. A real container should probably allow optional padding around the container and spacing between the child actors. You might also want to allow some child actors to expand to fill the available space, or align differently inside the container.

**Figure A-2. Behaviour**



Source Code ([./../examples/custom\\_container](#))

File: `examplebox.h`

```
#ifndef __EXAMPLE_BOX_H__
```

```

#define __EXAMPLE_BOX_H__

#include <clutter/clutter-actor.h>
#include <clutter/clutter-types.h>

G_BEGIN_DECLS

#define EXAMPLE_TYPE_BOX (example_box_get_type ())
#define EXAMPLE_BOX(obj) (G_TYPE_CHECK_INSTANCE_CAST ((obj), EXAMPLE_TYPE_BOX, ExampleBox))
#define EXAMPLE_IS_BOX(obj) (G_TYPE_CHECK_INSTANCE_TYPE ((obj), EXAMPLE_TYPE_BOX))
#define EXAMPLE_BOX_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST ((klass), EXAMPLE_TYPE_BOX, ExampleBoxClass))
#define EXAMPLE_IS_BOX_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE ((klass), EXAMPLE_TYPE_BOX))
#define EXAMPLE_BOX_GET_CLASS(obj) (G_TYPE_INSTANCE_GET_CLASS ((obj), EXAMPLE_TYPE_BOX, ExampleBoxClass))

typedef struct _ExampleBoxChild ExampleBoxChild;
typedef struct _ExampleBox ExampleBox;
typedef struct _ExampleBoxClass ExampleBoxClass;

struct _ExampleBox
{
    /*< private >*/
    ClutterActor parent_instance;

    /* Allocation of the box */
    ClutterActorBox allocation;

    /* List of ExampleBoxChild structures */
    GList *children;
};

struct _ExampleBoxClass
{
    /*< private >*/
    ClutterActorClass parent_class;
};

GType example_box_get_type (void) G_GNUC_CONST;

ClutterActor *example_box_new (void);
void example_box_pack (ExampleBox*box, ClutterActor *actor);
void example_box_remove_all (ExampleBox *box);

G_END_DECLS

#endif /* __EXAMPLE_BOX_H__ */

```

**File:** main.c

```

#include <clutter/clutter.h>
#include "examplebox.h"

```

```

#include <stdlib.h>

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor actor_color = { 0xff, 0xff, 0xff, 0x99 };
    ClutterColor actor_color2 = { 0x10, 0x40, 0x90, 0xff };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add our custom container to the stage: */
    ClutterActor *box = example_box_new ();
    clutter_actor_set_size (box, 100, 100);
    clutter_actor_set_position (box, 20, 20);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), box);
    clutter_actor_show (box);

    /* Add some actors to our container: */
    ClutterActor *actor = clutter_rectangle_new_with_color (&actor_color);
    clutter_actor_set_size (actor, 75, 75);
    clutter_container_add_actor (CLUTTER_CONTAINER (box), actor);
    clutter_actor_show (actor);

    ClutterActor *actor2 = clutter_rectangle_new_with_color (&actor_color2);
    clutter_actor_set_size (actor2, 75, 75);
    clutter_container_add_actor (CLUTTER_CONTAINER (box), actor2);
    clutter_actor_show (actor2);

    /* Show the stage: */
    clutter_actor_show (stage);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();

    return EXIT_SUCCESS;
}

```

**File:** examplebox.c

```

#include "clutter/cogl.h" /* For some helper functions. */

#include "examplebox.h"
#include <clutter/clutter-container.h>

#include <string.h>

```

```

/**
 * SECTION:example-box
 * @short_description: Simple example of a container actor.
 *
 * #ExampleBox imposes a specific layout on its children,
 * unlike #ClutterGroup which is a free-form container.
 *
 * Specifically, ExampleBox lays out its children along an imaginary
 * horizontal line.
 */

static void
layout_children (ExampleBox *box);

static void clutter_container_iface_init (ClutterContainerIface *iface);

G_DEFINE_TYPE_WITH_CODE (ExampleBox,
                        example_box,
                        CLUTTER_TYPE_ACTOR,
                        G_IMPLEMENT_INTERFACE (CLUTTER_TYPE_CONTAINER,
                                             clutter_container_iface_init));

/* An implementation for the ClutterContainer::add() vfunc: */
static void
example_box_add (ClutterContainer *container,
                ClutterActor      *actor)
{
    example_box_pack (EXAMPLE_BOX (container), actor);
}

/* An implementation for the ClutterContainer::remove() vfunc: */
static void
example_box_remove (ClutterContainer *container,
                   ClutterActor      *actor)
{
    ExampleBox *box = EXAMPLE_BOX (container);
    GList *l;

    g_object_ref (actor);

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        if (child == actor)
        {
            clutter_actor_unparent (child);

            box->children = g_list_remove_link (box->children, l);
            g_list_free (l);
        }
    }
}

```

```

        g_signal_emit_by_name (container, "actor-removed", actor);

        layout_children (box);

        if (CLUTTER_ACTOR_IS_VISIBLE (CLUTTER_ACTOR (box)))
            clutter_actor_queue_redraw (CLUTTER_ACTOR (box));

        break;
    }
}

g_object_unref (actor);
}

/* An implementation for the ClutterContainer::foreach() vfunc: */
static void
example_box_foreach (ClutterContainer *container,
                    ClutterCallback  callback,
                    gpointer           user_data)
{
    ExampleBox *box = EXAMPLE_BOX (container);
    GList *l;

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        (* callback) (child, user_data);
    }
}

static void
clutter_container_iface_init (ClutterContainerIface *iface)
{
    /* Provide implementations for ClutterContainer vfuncs: */
    iface->add = example_box_add;
    iface->remove = example_box_remove;
    iface->foreach = example_box_foreach;
}

/* An implementation for the ClutterActor::show_all() vfunc,
   showing all the child actors: */
static void
example_box_show_all (ClutterActor *actor)
{
    ExampleBox *box = EXAMPLE_BOX (actor);
    GList *l;

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

```

```

        clutter_actor_show (child);
    }

    clutter_actor_show (actor);
}

/* An implementation for the ClutterActor::hide_all() vfunc,
   hiding all the child actors: */
static void
example_box_hide_all (ClutterActor *actor)
{
    ExampleBox *box = EXAMPLE_BOX (actor);
    GList *l;

    clutter_actor_hide (actor);

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        clutter_actor_hide (child);
    }
}

/* An implementation for the ClutterActor::paint() vfunc,
   painting all the child actors: */
static void
example_box_paint (ClutterActor *actor)
{
    ExampleBox *box = EXAMPLE_BOX (actor);
    GList *l;

    cogl_push_matrix ();

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        if (CLUTTER_ACTOR_IS_MAPPED (child))
            clutter_actor_paint (child);
    }

    cogl_pop_matrix ();
}

/* An implementation for the ClutterActor::pick() vfunc,
   picking all the child actors: */
static void
example_box_pick (ClutterActor *actor,
                  const ClutterColor *color)
{
    ExampleBox *box = EXAMPLE_BOX (actor);
    GList *l;

```

```

for (l = box->children; l; l = l->next)
{
    ClutterActor *child = l->data;

    if (CLUTTER_ACTOR_IS_MAPPED (child))
        clutter_actor_pick (child, color);
}

/* An implementation for the ClutterActor::query_coords() vfunc: */
static void
example_box_query_coords (ClutterActor *actor,
                          ClutterActorBox *coords)
{
    ExampleBox *box = EXAMPLE_BOX (actor);
    GList *l;
    gint width, height;

    /* For this container,
     * x1 and y1 will just be whatever was provided to request_coords(),
     * but the (desired) height and width (via x2 and y2) will be based
     * on the height and width desired by the child actors.
     * Other containers might instead decide to reduce the size of the child actors
     * to fit inside the container's specified height/width.
     */
    coords->x1 = box->allocation.x1;
    coords->y1 = box->allocation.y1;

    /* Calculate the x2 and y2 allocation for this container,
     * based on the allocations requested by the children: */
    width = 0;
    height = 0;

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        if (CLUTTER_ACTOR_IS_VISIBLE (child))
        {
            guint child_width, child_height;

            clutter_actor_get_size (child, &child_width, &child_height);

            width = width + child_width;

            height = MAX (child_height, height);
        }
    }

    box->allocation.x2 = coords->x2 =
        coords->x1 + CLUTTER_UNITS_FROM_INT (width);
    box->allocation.y2 = coords->y2 =

```

```

        coords->y1 + CLUTTER_UNITS_FROM_INT (height);
    }

/* An implementation for the ClutterActor::request_coords() vfunc: */
static void
example_box_request_coords (ClutterActor *actor,
                           ClutterActorBox *coords)
{
    ExampleBox *box = EXAMPLE_BOX (actor);

    /* Store the provided allocation.
       But we only store x1 and y1 because the width and height are
       dependent on on the children. */
    box->allocation.x1 = coords->x1;
    box->allocation.y1 = coords->y1;
    box->allocation.x2 = -1;
    box->allocation.y2 = -1;

    /* Make sure that the children adapt their positions: */
    layout_children (EXAMPLE_BOX (actor));
}

static void
example_box_dispose (GObject *gobject)
{
    /* Destroy each child actor when this container is destroyed: */
    ExampleBox *box = EXAMPLE_BOX (gobject);
    GList *l;

    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        clutter_actor_destroy (child);
    }

    g_list_free (box->children);
    box->children = NULL;

    G_OBJECT_CLASS (example_box_parent_class)->dispose (gobject);
}

static void
example_box_class_init (ExampleBoxClass *klass)
{
    GObjectClass *gobject_class = G_OBJECT_CLASS (klass);
    ClutterActorClass *actor_class = CLUTTER_ACTOR_CLASS (klass);

    gobject_class->dispose = example_box_dispose;

    /* Provide implementations for ClutterActor vfuncs: */
    actor_class->show_all = example_box_show_all;
}

```

```

actor_class->hide_all = example_box_hide_all;
actor_class->paint = example_box_paint;
actor_class->pick = example_box_pick;
actor_class->query_coords = example_box_query_coords;
actor_class->request_coords = example_box_request_coords;
}

static void
example_box_init (ExampleBox *box)
{
    box->allocation.x1 = box->allocation.y1 = 0;
    box->allocation.x2 = box->allocation.y2 = -1;
}

static void
layout_children (ExampleBox *box)
{
    /* Get the size requested by this container: */
    ClutterActorBox allocation = { 0, };
    clutter_actor_query_coords (CLUTTER_ACTOR (box), &allocation);

    ClutterUnit width = allocation.x2 - allocation.x1;
    ClutterUnit height = allocation.y2 - allocation.y1;

    if (width < 0)
        width = 0;

    if (height < 0)
        height = 0;

    /* Look at each child actor: */
    ClutterUnit child_x = 0;
    GList *l = NULL;
    for (l = box->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        /* Discover what size the child wants: */
        ClutterActorBox child_req = { 0, };
        clutter_actor_query_coords (child, &child_req);

        const ClutterUnit child_width = child_req.x2 - child_req.x1;
        const ClutterUnit child_height = child_req.y2 - child_req.y1;

        /* Calculate the position and size that the child may actually have: */

        /* Position the child just after the previous child, horizontally: */
        ClutterActorBox child_box = { 0, };
        child_box.x1 = child_x;
        child_box.x2 = child_x + child_width;
        child_x = child_box.x2;

```

```

    /* Position the child at the top of the container: */
    child_box.y1 = 0;
    child_box.y2 = child_height;

    /* Tell the child what position and size it may actually have: */
    clutter_actor_request_coords (child, &child_box);
}
}

/*
 * Public API
 */

/**
 * example_box_pack:
 * @box: a #ExampleBox
 * @actor: a #ClutterActor to pack into the box
 *
 * Packs @actor into @box.
 */
void
example_box_pack (ExampleBox      *box,
                  ClutterActor    *actor)
{
    g_return_if_fail (EXAMPLE_IS_BOX (box));
    g_return_if_fail (CLUTTER_IS_ACTOR (actor));

    box->children = g_list_prepend (box->children, actor);
    clutter_actor_set_parent (actor, CLUTTER_ACTOR (box));

    /* Reset the saved allocation,
     * so that it will be recalculated: */
    box->allocation.x2 = box->allocation.y2 = -1;

    layout_children (EXAMPLE_BOX (box));

    if (CLUTTER_ACTOR_IS_VISIBLE (CLUTTER_ACTOR (box)))
        clutter_actor_queue_redraw (CLUTTER_ACTOR (box));
}

/**
 * example_box_remove_all:
 * @box: a #ExampleBox
 *
 * Removes all child actors from the #ExampleBox
 */
void
example_box_remove_all (ExampleBox *box)
{
    GList *children;

```

```
g_return_if_fail (EXAMPLE_IS_BOX (box));

children = box->children;
while (children)
{
    ClutterActor *child = children->data;
    children = children->next;

    clutter_container_remove_actor (CLUTTER_CONTAINER (box), child);
}

/**
 * example_box_new:
 *
 * Creates a new box.
 *
 * Return value: the newly created #ExampleBox
 */
ClutterActor *
example_box_new (void)
{
    return g_object_new (EXAMPLE_TYPE_BOX, NULL);
}
```

# Appendix B. Implementing Scrolling in a Window-like Actor

## B.1. The Technique

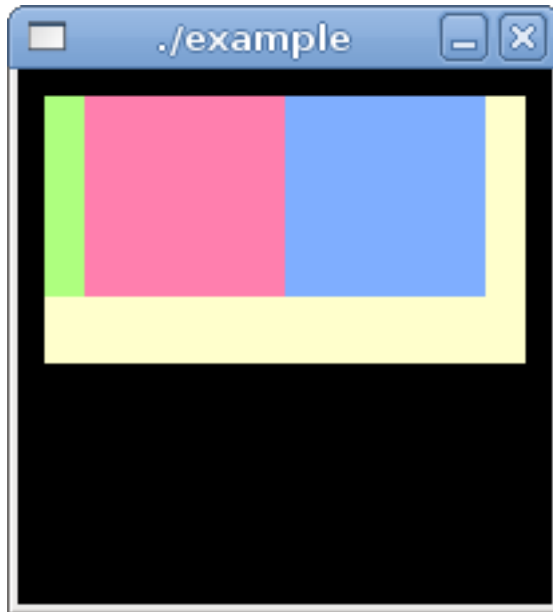
There is not yet a standard container in Clutter to show and scroll through a small part of a set of widgets, like the `GtkScrolledWindow` widget in the GTK+ toolkit, so you may need to implement this functionality in your application.

The Tidy project contains some suggested implementations for such actors, but here is a simpler example of the general technique. It creates the impression of scrolling by clipping a container so that it only shows a small area of the screen, while moving the child widgets in that container.

## B.2. Example

This example places three rectangles in a custom container which scrolls its child widgets to the left when the user clicks on the stage.

Real-world applications will of course want to implement more specific behaviour, depending on their needs. For instance, adding scrollbars that show accurate scroll positions, scrolling smoothly with animation, efficiently drawing only objects that should be visible when dealing with large numbers of rows.

**Figure B-1. Scrolling Container**

Source Code (`../../examples/scrolling`)

File: `scrollingcontainer.h`

```

#ifndef __EXAMPLE_SCROLLING_CONTAINER_H__
#define __EXAMPLE_SCROLLING_CONTAINER_H__

#include <clutter/clutter-actor.h>
#include <clutter/clutter-types.h>

G_BEGIN_DECLS

#define EXAMPLE_TYPE_SCROLLING_CONTAINER (example_scrolling_container_get_type())
#define EXAMPLE_SCROLLING_CONTAINER(obj) (G_TYPE_CHECK_INSTANCE_CAST ((obj),
#define EXAMPLE_IS_SCROLLING_CONTAINER(obj) (G_TYPE_CHECK_INSTANCE_TYPE ((obj),
#define EXAMPLE_SCROLLING_CONTAINER_CLASS(klass) (G_TYPE_CHECK_CLASS_CAST ((klass),
#define EXAMPLE_IS_SCROLLING_CONTAINER_CLASS(klass) (G_TYPE_CHECK_CLASS_TYPE ((klass),
#define EXAMPLE_SCROLLING_CONTAINER_GET_CLASS(obj) (G_TYPE_INSTANCE_GET_CLASS ((obj),

typedef struct _ExampleScrollingContainerChild ExampleScrollingContainerChild;
typedef struct _ExampleScrollingContainer ExampleScrollingContainer;
typedef struct _ExampleScrollingContainerClass ExampleScrollingContainerClass;

struct _ExampleScrollingContainer

```

```

{
  /*< private >*/
  ClutterActor parent_instance;

  /* Allocation of the container: */
  ClutterActorBox allocation;

  /* List of ExampleScrollingContainerChild structures */
  GList *children;

  /* All the child actors are in this group: */
  ClutterActor *group;
  gint offset;

  /* A rectangle to show the bounds: */
  ClutterActor *rect;
};

struct _ExampleScrollingContainerClass
{
  /*< private >*/
  ClutterActorClass parent_class;
};

GType example_scrolling_container_get_type (void) G_GNUC_CONST;

ClutterActor *example_scrolling_container_new (void);
void example_scrolling_container_pack (ExampleScrollingContainer *self, ClutterActor *actor);
void example_scrolling_container_remove_all (ExampleScrollingContainer *self);

void example_scrolling_container_scroll_left (ExampleScrollingContainer *self, gint distance);

G_END_DECLS

#endif /* __EXAMPLE_SCROLLING_CONTAINER_H__ */

```

File: main.c

```

#include <clutter/clutter.h>
#include "scrollingcontainer.h"
#include <stdlib.h>

ClutterActor *scrolling = NULL;

static gboolean
on_stage_button_press (ClutterStage *stage, ClutterEvent *event, gpointer data)
{
  printf ("Scrolling\n");

  /* Scroll the container: */

```

## Appendix B. Implementing Scrolling in a Window-like Actor

```
example_scrolling_container_scroll_left (
    EXAMPLE_SCROLLING_CONTAINER (scrolling), 10);

return TRUE; /* Stop further handling of this event. */
}

int main(int argc, char *argv[])
{
    ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
    ClutterColor actor_color = { 0x7f, 0xae, 0xff, 0xff };
    ClutterColor actor_color2 = { 0xff, 0x7f, 0xae, 0xff };
    ClutterColor actor_color3 = { 0xae, 0xff, 0x7f, 0xff };

    clutter_init (&argc, &argv);

    /* Get the stage and set its size and color: */
    ClutterActor *stage = clutter_stage_get_default ();
    clutter_actor_set_size (stage, 200, 200);
    clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

    /* Add our scrolling container to the stage: */
    scrolling = example_scrolling_container_new ();
    clutter_actor_set_size (scrolling, 180, 100);
    clutter_actor_set_position (scrolling, 10, 10);
    clutter_container_add_actor (CLUTTER_CONTAINER (stage), scrolling);
    clutter_actor_show (scrolling);

    /* Add some actors to our container: */
    ClutterActor *actor = clutter_rectangle_new_with_color (&actor_color);
    clutter_actor_set_size (actor, 75, 75);
    clutter_container_add_actor (CLUTTER_CONTAINER (scrolling), actor);
    clutter_actor_show (actor);

    ClutterActor *actor2 = clutter_rectangle_new_with_color (&actor_color2);
    clutter_actor_set_size (actor2, 75, 75);
    clutter_container_add_actor (CLUTTER_CONTAINER (scrolling), actor2);
    clutter_actor_show (actor2);

    ClutterActor *actor3 = clutter_rectangle_new_with_color (&actor_color3);
    clutter_actor_set_size (actor3, 75, 75);
    clutter_container_add_actor (CLUTTER_CONTAINER (scrolling), actor3);
    clutter_actor_show (actor3);

    /* Show the stage: */
    clutter_actor_show (stage);

    /* Connect signal handlers to handle mouse clicks on the stage: */
    g_signal_connect (stage, "button-press-event",
        G_CALLBACK (on_stage_button_press), NULL);

    /* Start the main loop, so we can respond to events: */
    clutter_main ();
}
```

```
    return EXIT_SUCCESS;
}
```

**File:** scrollingcontainer.c

```
#include "clutter/cogl.h" /* For some helper functions. */

#include "scrollingcontainer.h"
#include <clutter/clutter-container.h>
#include <clutter/clutter-group.h>
#include <clutter/clutter-rectangle.h>

#include <string.h>
#include <stdio.h>

/**
 * SECTION:example-scrolling-container
 * @short_description: This container shows only a small area
 * of its child actors, and the child actors can be scrolled
 * left under that area.
 */

static void
layout_children (ExampleScrollingContainer *self);

static void clutter_container_iface_init (ClutterContainerIface *iface);

G_DEFINE_TYPE_WITH_CODE (ExampleScrollingContainer,
                        example_scrolling_container,
                        CLUTTER_TYPE_ACTOR,
                        G_IMPLEMENT_INTERFACE (CLUTTER_TYPE_CONTAINER,
                                              clutter_container_iface_init));

/* An implementation for the ClutterContainer::add() vfunc: */
static void
example_scrolling_container_add (ClutterContainer *container,
                                ClutterActor      *actor)
{
    example_scrolling_container_pack (EXAMPLE_SCROLLING_CONTAINER (container), actor);
}

/* An implementation for the ClutterContainer::remove() vfunc: */
static void
example_scrolling_container_remove (ClutterContainer *container,
                                    ClutterActor      *actor)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (container);
```

```

GList *l;

g_object_ref (actor);

for (l = self->children; l; l = l->next)
{
    ClutterActor *child = l->data;

    if (child == actor)
    {
        clutter_container_remove_actor (CLUTTER_CONTAINER (self->group), child);

        self->children = g_list_remove_link (self->children, l);
        g_list_free (l);

        g_signal_emit_by_name (container, "actor-removed", actor);

        layout_children (self);

        if (CLUTTER_ACTOR_IS_VISIBLE (CLUTTER_ACTOR (self)))
            clutter_actor_queue_redraw (CLUTTER_ACTOR (self));

        break;
    }
}

g_object_unref (actor);
}

/* An implementation for the ClutterContainer::foreach() vfunc: */
static void
example_scrolling_container_foreach (ClutterContainer *container,
                                     ClutterCallback  callback,
                                     gpointer           user_data)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (container);
    clutter_container_foreach (CLUTTER_CONTAINER (self->group), callback, user_data);
}

static void
clutter_container_iface_init (ClutterContainerIface *iface)
{
    /* Provide implementations for ClutterContainer vfuncs: */
    iface->add = example_scrolling_container_add;
    iface->remove = example_scrolling_container_remove;
    iface->foreach = example_scrolling_container_foreach;
}

/* An implementation for the ClutterActor::show_all() vfunc,
   showing all the child actors: */
static void
example_scrolling_container_show_all (ClutterActor *actor)
{

```

## Appendix B. Implementing Scrolling in a Window-like Actor

```
ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (actor);
GList *l;

for (l = self->children; l; l = l->next)
{
    ClutterActor *child = l->data;

    clutter_actor_show (child);
}

clutter_actor_show (actor);
}

/* An implementation for the ClutterActor::hide_all() vfunc,
   hiding all the child actors: */
static void
example_scrolling_container_hide_all (ClutterActor *actor)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (actor);
    GList *l;

    clutter_actor_hide (actor);

    for (l = self->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        clutter_actor_hide (child);
    }
}

/* An implementation for the ClutterActor::paint() vfunc,
   painting all the child actors: */
static void
example_scrolling_container_paint (ClutterActor *actor)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (actor);
    clutter_actor_paint (self->group);
}

/* An implementation for the ClutterActor::pick() vfunc,
   picking all the child actors: */
static void
example_scrolling_container_pick (ClutterActor *actor,
                                 const ClutterColor *color)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (actor);
    clutter_actor_pick (self->group, color);
}

/* An implementation for the ClutterActor::query_coords() vfunc: */
static void
example_scrolling_container_query_coords (ClutterActor *actor,
```

## Appendix B. Implementing Scrolling in a Window-like Actor

```
        ClutterActorBox *coords)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (actor);

    /* The allocation is whatever was set with
     * clutter_actor_set_width(), clutter_actor_set_height(), etc.
     */
    coords->x1 = self->allocation.x1;
    coords->y1 = self->allocation.y1;
    coords->x2 = self->allocation.x2;
    coords->y2 = self->allocation.y2;
}

/* An implementation for the ClutterActor::request_coords() vfunc: */
static void
example_scrolling_container_request_coords (ClutterActor      *actor,
        ClutterActorBox *coords)
{
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (actor);

    /* Store the provided allocation:*/
    self->allocation.x1 = coords->x1;
    self->allocation.y1 = coords->y1;
    self->allocation.x2 = coords->x2;
    self->allocation.y2 = coords->y2;

    /* Make sure that the children adapt their positions: */
    layout_children (EXAMPLE_SCROLLING_CONTAINER (actor));
}

static void
example_scrolling_container_dispose (GObject *gobject)
{
    /* Destroy each child actor when this container is destroyed: */
    ExampleScrollingContainer *self = EXAMPLE_SCROLLING_CONTAINER (gobject);
    GList *l;

    for (l = self->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        clutter_actor_destroy (child);
    }

    g_list_free (self->children);
    self->children = NULL;

    if (self->group)
    {
        clutter_actor_destroy (self->group);
        self->group = NULL;
    }
}
```

## Appendix B. Implementing Scrolling in a Window-like Actor

```
G_OBJECT_CLASS (example_scrolling_container_parent_class)->dispose (gobject);
}

static void
example_scrolling_container_class_init (ExampleScrollingContainerClass *klass)
{
    GObjectClass *gobject_class = G_OBJECT_CLASS (klass);
    ClutterActorClass *actor_class = CLUTTER_ACTOR_CLASS (klass);

    gobject_class->dispose = example_scrolling_container_dispose;

    /* Provide implementations for ClutterActor vfuncs: */
    actor_class->show_all = example_scrolling_container_show_all;
    actor_class->hide_all = example_scrolling_container_hide_all;
    actor_class->paint = example_scrolling_container_paint;
    actor_class->pick = example_scrolling_container_pick;
    actor_class->query_coords = example_scrolling_container_query_coords;
    actor_class->request_coords = example_scrolling_container_request_coords;
}

static void
example_scrolling_container_init (ExampleScrollingContainer *self)
{
    self->group = clutter_group_new ();
    clutter_actor_show (self->group);
    self->offset = 0;

    /* A rectange to show the bounds: */
    ClutterColor actor_color = { 0xff, 0xff, 0xcc, 0xff };
    self->rect = clutter_rectangle_new_with_color (&actor_color);
    clutter_container_add_actor (CLUTTER_CONTAINER (self->group), self->rect);
    clutter_actor_show (self->rect);

    self->allocation.x1 = self->allocation.y1 = 99;
    self->allocation.x2 = self->allocation.y2 = 99;
}

static void
layout_children (ExampleScrollingContainer *self)
{
    /* Get the size requested by this container: */
    ClutterActorBox allocation = { 0, };
    clutter_actor_query_coords (CLUTTER_ACTOR (self), &allocation);

    ClutterUnit width = allocation.x2 - allocation.x1;
    ClutterUnit height = allocation.y2 - allocation.y1;

    if (width < 0)
        width = 0;

    if (height < 0)
```

## Appendix B. Implementing Scrolling in a Window-like Actor

```
    height = 0;

    /* Arrange the group: */
    ClutterActorBox child_box = { 0, };
    child_box.x1 = 0;
    child_box.x2 = child_box.x1 + width;

    /* Position the child at the top of the container: */
    child_box.y1 = 0;
    child_box.y2 = height;

    clutter_actor_request_coords (self->group, &child_box);

    /* Make sure that the group only shows the specified area, by clipping: */
    clutter_actor_set_clip (self->group, 0, 0, CLUTTER_UNITS_TO_DEVICE(width), CLUTTER_UNITS_

    /* Show a rectangle border to show the area: */
    clutter_actor_set_position (self->rect, 0, 0);
    clutter_actor_set_width (self->rect, CLUTTER_UNITS_TO_DEVICE(width));
    clutter_actor_set_height (self->rect, CLUTTER_UNITS_TO_DEVICE(height));
    clutter_actor_lower (self->rect, NULL);

    /* Look at each child actor: */
    ClutterUnit child_x = -(self->offset);
    GList *l = NULL;
    for (l = self->children; l; l = l->next)
    {
        ClutterActor *child = l->data;

        gint child_width = clutter_actor_get_width (child);
        clutter_actor_set_position (child, child_x, 0);
        child_x += child_width;
    }
}

/*
 * Public API
 */

/**
 * example_scrolling_container_pack:
 * @self: a #ExampleScrollingContainer
 * @actor: a #ClutterActor to pack into the self
 *
 * Packs @actor into @self.
 */
void
example_scrolling_container_pack (ExampleScrollingContainer      *self,
                                ClutterActor                    *actor)
{

    g_return_if_fail (EXAMPLE_IS_SCROLLING_CONTAINER (self));
```

## Appendix B. Implementing Scrolling in a Window-like Actor

```
g_return_if_fail (CLUTTER_IS_ACTOR (actor));

self->children = g_list_prepend (self->children, actor);
clutter_container_add_actor (CLUTTER_CONTAINER (self->group), actor);

layout_children (EXAMPLE_SCROLLING_CONTAINER (self));

if (CLUTTER_ACTOR_IS_VISIBLE (CLUTTER_ACTOR (self)))
    clutter_actor_queue_redraw (CLUTTER_ACTOR (self));
}

/**
 * example_scrolling_container_remove_all:
 * @self: a #ExampleScrollingContainer
 *
 * Removes all child actors from the #ExampleScrollingContainer
 */
void
example_scrolling_container_remove_all (ExampleScrollingContainer *self)
{
    GList *children;

    g_return_if_fail (EXAMPLE_IS_SCROLLING_CONTAINER (self));

    children = self->children;
    while (children)
    {
        ClutterActor *child = children->data;
        children = children->next;

        clutter_container_remove_actor (CLUTTER_CONTAINER (self), child);
    }
}

/**
 * example_scrolling_container_new:
 *
 * Creates a new self.
 *
 * Return value: the newly created #ExampleScrollingContainer
 */
ClutterActor *
example_scrolling_container_new (void)
{
    return g_object_new (EXAMPLE_TYPE_SCROLLING_CONTAINER, NULL);
}

/**
 * example_scrolling_container_scroll_left:
 * @self: a #ExampleScrollingContainer
 * @distance: The number of pixels by which to scroll left.

```

## *Appendix B. Implementing Scrolling in a Window-like Actor*

```
*
* Scroll all the child widgets left,
* resulting in some parts being hidden,
* and some parts becoming visible.
*/
void example_scrolling_container_scroll_left (ExampleScrollingContainer *self, gint distance)
{
    g_return_if_fail (EXAMPLE_IS_SCROLLING_CONTAINER (self));

    self->offset += distance;
    layout_children (self);
}
```

# Appendix C. Implementing Text Editing

## C.1. The Technique

Clutter provides a `ClutterEntry` actor for text entry. However, this is limited to single lines of text. Therefore you will need to implement your own custom actor if you need this functionality with multiple lines of text.

Like `ClutterEntry` you can use the Pango API - specifically the `PangoLayout` object. The `PangoLayout` can then be rendered to the clutter display in your `ClutterActor::paint()` implementation by using the `pango_clutter_render_layout()` utility function.

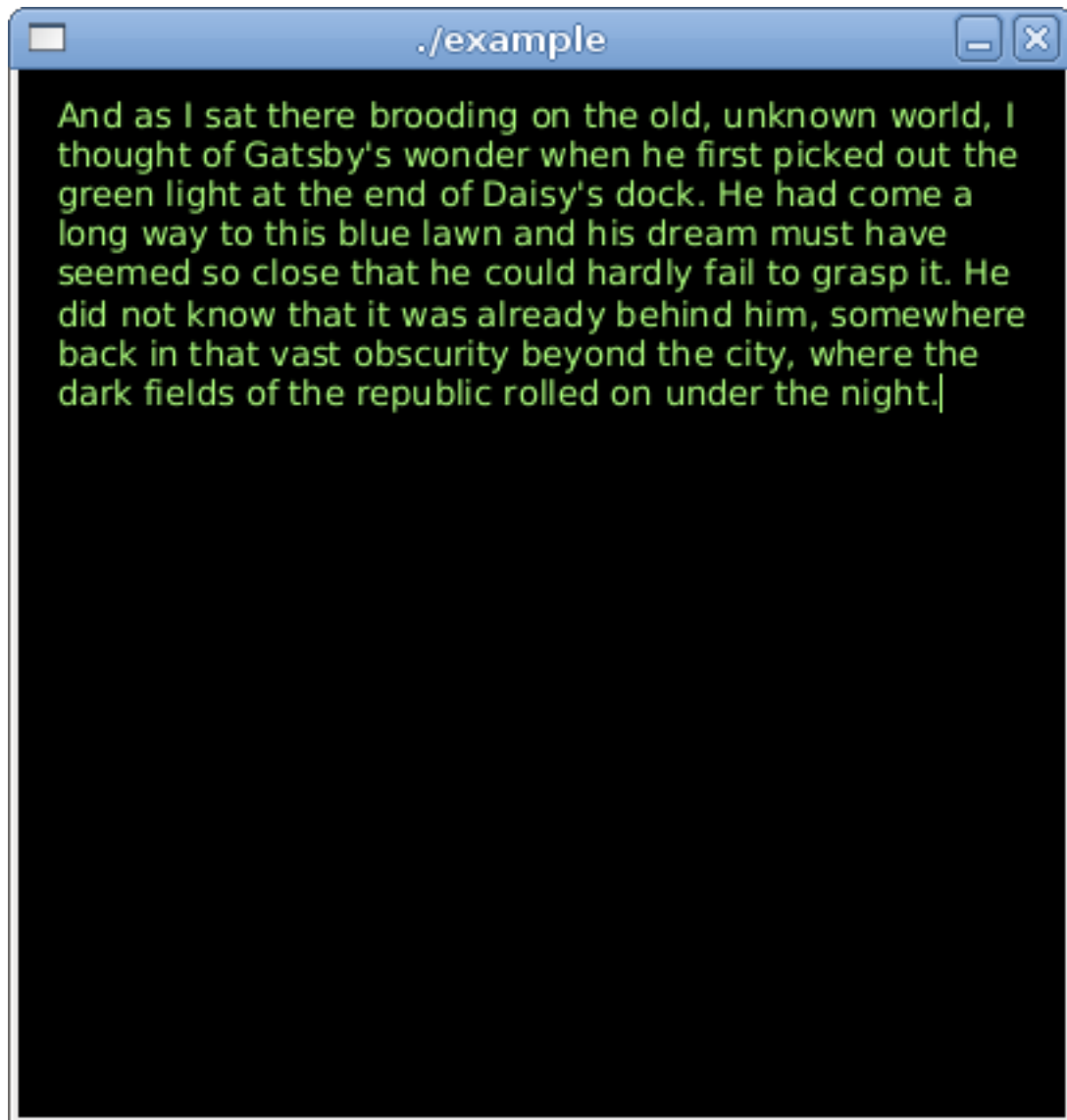
**Note:** However, `pango_clutter_render_layout()` is not official public Clutter API and could disappear in future version of Clutter. You may choose to investigate its implementation and reimplement it in your application. Future versions of Pango are likely to provide new API to make this easier.

## C.2. Example

This example has a custom actor, based on the `ClutterEntry` implementation, using a `PangoLayout` to show text wrapped over multiple lines.

Real-world applications will probably want to implement more text-editing features, such as the ability to move the cursor vertically, the ability to select and copy sections of text, the ability to show and manipulate right-to-left text, etc.

Figure C-1. Multiline Text Entry



Source Code (`../../examples/multiline_text_entry`)

File: `multiline_entry.h`

```
#ifndef _HAVE_EXAMPLE_MULTILINE_ENTRY_H
#define _HAVE_EXAMPLE_MULTILINE_ENTRY_H

#include <clutter/clutter-actor.h>
```

```

#include <clutter/clutter-color.h>
#include <clutter/clutter-event.h>
#include <pango/pango.h>

G_BEGIN_DECLS

#define CLUTTER_TYPE_MULTILINE_ENTRY (example_multiline_entry_get_type ())

#define EXAMPLE_MULTILINE_ENTRY(obj) \
    (G_TYPE_CHECK_INSTANCE_CAST ((obj), \
    CLUTTER_TYPE_MULTILINE_ENTRY, ExampleMultilineEntry))

#define EXAMPLE_MULTILINE_ENTRY_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_CAST ((klass), \
    CLUTTER_TYPE_MULTILINE_ENTRY, ExampleMultilineEntryClass))

#define EXAMPLE_IS_MULTILINE_ENTRY(obj) \
    (G_TYPE_CHECK_INSTANCE_TYPE ((obj), \
    CLUTTER_TYPE_MULTILINE_ENTRY))

#define EXAMPLE_IS_MULTILINE_ENTRY_CLASS(klass) \
    (G_TYPE_CHECK_CLASS_TYPE ((klass), \
    CLUTTER_TYPE_MULTILINE_ENTRY))

#define EXAMPLE_MULTILINE_ENTRY_GET_CLASS(obj) \
    (G_TYPE_INSTANCE_GET_CLASS ((obj), \
    CLUTTER_TYPE_MULTILINE_ENTRY, ExampleMultilineEntryClass))

typedef struct _ExampleMultilineEntry ExampleMultilineEntry;
typedef struct _ExampleMultilineEntryClass ExampleMultilineEntryClass;
typedef struct _ExampleMultilineEntryPrivate ExampleMultilineEntryPrivate;

struct _ExampleMultilineEntry
{
    /*< private >*/
    ClutterActor parent_instance;

    ExampleMultilineEntryPrivate *priv;
};

/**
 * ExampleMultilineEntryClass:
 * @paint_cursor: virtual function for subclasses to use to draw a custom
 * cursor instead of the default one
 * @text_changed: signal class handler for ExampleMultilineEntry::text-changed
 * @cursor_event: signal class handler for ExampleMultilineEntry::cursor-event
 * @activate: signal class handler for ExampleMultilineEntry::activate
 */
struct _ExampleMultilineEntryClass
{
    /*< private >*/
    ClutterActorClass parent_class;

```

```

/*< public >*/
/* vfuncs, not signals */
void (* paint_cursor) (ExampleMultilineEntry *entry);

/* signals */
void (* text_changed) (ExampleMultilineEntry *entry);
void (* cursor_event) (ExampleMultilineEntry *entry,
                      ClutterGeometry *geometry);
void (* activate) (ExampleMultilineEntry *entry);
};

GType example_multiline_entry_get_type (void) G_GNUC_CONST;

ClutterActor *      example_multiline_entry_new          (void);
void                example_multiline_entry_set_text    (ExampleMultilineEntry
                const gchar *text);
G_CONST_RETURN gchar *example_multiline_entry_get_text  (ExampleMultilineEntry
void                example_multiline_entry_set_font_name (ExampleMultilineEntry
                const gchar *font_name);
G_CONST_RETURN gchar *example_multiline_entry_get_font_name (ExampleMultilineEntry
void                example_multiline_entry_set_color    (ExampleMultilineEntry
                const ClutterColor *color);
void                example_multiline_entry_get_color    (ExampleMultilineEntry
                ClutterColor *color);

void                example_multiline_entry_handle_key_event (ExampleMultilineEntry
                ClutterKeyEvent *kev);

G_END_DECLS

#endif /* _HAVE_EXAMPLE_MULTILINE_ENTRY_H */

```

**File:** main.c

```

#include <clutter/clutter.h>
#include "multiline_entry.h"
#include <stdlib.h>

ClutterActor *multiline = NULL;

static gboolean
on_stage_key_press (ClutterStage *stage, ClutterKeyEvent *event, gpointer data)
{
    /* TODO: Ideally the entry would handle this itself. */
    example_multiline_entry_handle_key_event (
        EXAMPLE_MULTILINE_ENTRY (multiline), event);

    return TRUE; /* Stop further handling of this event. */
}

int main(int argc, char *argv[])

```

```

{
ClutterColor stage_color = { 0x00, 0x00, 0x00, 0xff };
ClutterColor actor_color = { 0xae, 0xff, 0x7f, 0xff };

clutter_init (&argc, &argv);

/* Get the stage and set its size and color: */
ClutterActor *stage = clutter_stage_get_default ();
clutter_actor_set_size (stage, 400, 400);
clutter_stage_set_color (CLUTTER_STAGE (stage), &stage_color);

/* Add our multiline entry to the stage: */
multiline = example_multiline_entry_new ();
example_multiline_entry_set_text (EXAMPLE_MULTILINE_ENTRY (multiline),
    "And as I sat there brooding on the old, unknown world, I thought of "
    "Gatsby's wonder when he first picked out the green light at the end of "
    "Daisy's dock. He had come a long way to this blue lawn and his dream "
    "must have seemed so close that he could hardly fail to grasp it. He did "
    "not know that it was already behind him, somewhere back in that vast "
    "obscurity beyond the city, where the dark fields of the republic rolled "
    "on under the night.");
example_multiline_entry_set_color (EXAMPLE_MULTILINE_ENTRY (multiline),
    &actor_color);
clutter_actor_set_size (multiline, 380, 380);
clutter_actor_set_position (multiline, 10, 10);
clutter_container_add_actor (CLUTTER_CONTAINER (stage), multiline);
clutter_actor_show (multiline);

/* Connect signal handlers to handle key presses on the stage: */
g_signal_connect (stage, "key-press-event",
    G_CALLBACK (on_stage_key_press), NULL);

/* Show the stage: */
clutter_actor_show (stage);

/* Start the main loop, so we can respond to events: */
clutter_main ();

return EXIT_SUCCESS;
}

```

**File:** multiline\_entry.c

```

#include "multiline_entry.h"

#include <clutter/clutter-enum-types.h>
#include <clutter/clutter-keysyms.h>
#include <clutter/clutter-main.h>
#include <clutter/clutter-rectangle.h>
#include <clutter/clutter-units.h>

```

```

#include <clutter/pangoclutter.h>
#include <clutter/clutter-backend.h>
#include <string.h>

#define DEFAULT_FONT_NAME "Sans 10"
#define ENTRY_CURSOR_WIDTH 1

G_DEFINE_TYPE (ExampleMultilineEntry, example_multiline_entry, CLUTTER_TYPE_ACTOR);

/* For the font map: */
static PangoClutterFontMap *_font_map = NULL;
static PangoContext *_context = NULL;

enum
{
    PROP_0,

    PROP_FONT_NAME,
    PROP_TEXT,
    PROP_COLOR
};

enum
{
    TEXT_CHANGED,
    CURSOR_EVENT,

    LAST_SIGNAL
};

static guint entry_signals[LAST_SIGNAL] = { 0, };

#define EXAMPLE_MULTILINE_ENTRY_GET_PRIVATE(obj) \
(G_TYPE_INSTANCE_GET_PRIVATE ((obj), CLUTTER_TYPE_MULTILINE_ENTRY, ExampleMultilineEntryPrivate))

static void
example_multiline_entry_delete_chars (ExampleMultilineEntry *entry, guint num);

static void
example_multiline_entry_insert_unichar (ExampleMultilineEntry *entry, gunichar wc);

static void
example_multiline_entry_delete_text (ExampleMultilineEntry *entry, gssize start_pos, gssize end_pos);

struct _ExampleMultilineEntryPrivate
{
    PangoContext *_context;
    PangoFontDescription *desc;

    ClutterColor fgcol;

    gchar *text;
    gchar *font_name;
};

```

```

gint          width;
gint          n_chars;

gint          position;
gint          text_x;

PangoAttrList *effective_attrs;
PangoLayout   *layout;

ClutterGeometry cursor_pos;
ClutterActor   *cursor;
};

static void
example_multiline_entry_set_property (GObject      *object,
    guint      prop_id,
    const GValue *value,
    GParamSpec *pspec)
{
    ExampleMultilineEntry *entry;
    ExampleMultilineEntryPrivate *priv;

    entry = EXAMPLE_MULTILINE_ENTRY (object);
    priv = entry->priv;

    switch (prop_id)
    {
        case PROP_FONT_NAME:
            example_multiline_entry_set_font_name (entry, g_value_get_string (value));
            break;
        case PROP_TEXT:
            example_multiline_entry_set_text (entry, g_value_get_string (value));
            break;
        case PROP_COLOR:
            example_multiline_entry_set_color (entry, g_value_get_boxed (value));
            break;
        default:
            G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
            break;
    }
}

static void
example_multiline_entry_get_property (GObject      *object,
    guint      prop_id,
    GValue     *value,
    GParamSpec *pspec)
{
    ExampleMultilineEntry *entry;
    ExampleMultilineEntryPrivate *priv;
    ClutterColor          color;

```

```

entry = EXAMPLE_MULTILINE_ENTRY(object);
priv = entry->priv;

switch (prop_id)
{
case PROP_FONT_NAME:
    g_value_set_string (value, priv->font_name);
    break;
case PROP_TEXT:
    g_value_set_string (value, priv->text);
    break;
case PROP_COLOR:
    example_multiline_entry_get_color (entry, &color);
    g_value_set_boxed (value, &color);
    break;
default:
    G_OBJECT_WARN_INVALID_PROPERTY_ID (object, prop_id, pspec);
    break;
}
}

static void
example_multiline_entry_ensure_layout (ExampleMultilineEntry *entry, gint width)
{
    ExampleMultilineEntryPrivate *priv;

    priv = entry->priv;

    if (!priv->layout)
    {
        priv->layout = pango_layout_new (_context);

        if (priv->effective_attrs)
            pango_layout_set_attributes (priv->layout, priv->effective_attrs);

        pango_layout_set_single_paragraph_mode (priv->layout,
            FALSE );

        pango_layout_set_font_description (priv->layout, priv->desc);

        pango_layout_set_text (priv->layout, priv->text, priv->n_chars);

        pango_layout_set_wrap (priv->layout, PANGO_WRAP_WORD);

        if (width > 0)
            pango_layout_set_width (priv->layout, width * PANGO_SCALE);
        else
            pango_layout_set_width (priv->layout, -1);
    }
}

static void

```

```

example_multiline_entry_clear_layout (ExampleMultilineEntry *entry)
{
    if (entry->priv->layout)
    {
        g_object_unref (entry->priv->layout);
        entry->priv->layout = NULL;
    }
}

static gint
offset_to_bytes (const gchar *text, gint pos)
{
    gchar *c = NULL;
    gint i, j, len;

    if (pos < 1)
        return pos;

    c = g_utf8_next_char (text);
    j = 1;
    len = strlen (text);

    for (i = 0; i < len; i++)
    {
        if (&text[i] == c)
        {
            if (j == pos)
                break;
            else
            {
                c = g_utf8_next_char (c);
                j++;
            }
        }
    }
    return i;
}

static void
example_multiline_entry_ensure_cursor_position (ExampleMultilineEntry *entry)
{
    ExampleMultilineEntryPrivate *priv;
    gint index_;
    PangoRectangle rect;

    priv = entry->priv;

    if (priv->position == -1)
        index_ = strlen (priv->text);
    else
        index_ = offset_to_bytes (priv->text, priv->position);
}

```

```

pango_layout_get_cursor_pos (priv->layout, index_, &rect, NULL);
priv->cursor_pos.x = rect.x / PANGO_SCALE;
priv->cursor_pos.y = rect.y / PANGO_SCALE;
priv->cursor_pos.width = ENTRY_CURSOR_WIDTH;
priv->cursor_pos.height = rect.height / PANGO_SCALE;

g_signal_emit (entry, entry_signals[CURSOR_EVENT], 0, &priv->cursor_pos);
}

static void
example_multiline_entry_clear_cursor_position (ExampleMultilineEntry *entry)
{
    entry->priv->cursor_pos.width = 0;
}

void
example_multiline_entry_paint_cursor (ExampleMultilineEntry *entry)
{
    ExampleMultilineEntryPrivate *priv;

    priv = entry->priv;

    clutter_actor_set_size (CLUTTER_ACTOR (priv->cursor),
                           priv->cursor_pos.width,
                           priv->cursor_pos.height);

    clutter_actor_set_position (priv->cursor,
                                priv->cursor_pos.x,
                                priv->cursor_pos.y);

    clutter_actor_paint (priv->cursor);
}

static void
example_multiline_entry_paint (ClutterActor *self)
{
    ExampleMultilineEntry *entry;
    ExampleMultilineEntryPrivate *priv;
    PangoRectangle logical;
    gint width, actor_width;
    gint text_width;
    gint cursor_x;

    entry = EXAMPLE_MULTILINE_ENTRY (self);
    priv = entry->priv;

    if (priv->desc == NULL || priv->text == NULL)
    {
        return;
    }

    if (priv->width < 0)
        width = clutter_actor_get_width (self);

```

```

else
    width = priv->width;

clutter_actor_set_clip (self, 0, 0,
                       width,
                       clutter_actor_get_height (self));

actor_width = width;
example_multiline_entry_ensure_layout (entry, actor_width);
example_multiline_entry_ensure_cursor_position (entry);

pango_layout_get_extents (priv->layout, NULL, &logical);
text_width = logical.width / PANGO_SCALE;

if (actor_width < text_width)
{
    /* We need to do some scrolling */
    cursor_x = priv->cursor_pos.x;

    /* If the cursor is at the beginning or the end of the text, the placement
     * is easy, however, if the cursor is in the middle somewhere, we need to
     * make sure the text doesn't move until the cursor is either in the
     * far left or far right
     */

    if (priv->position == 0)
        priv->text_x = 0;
    else if (priv->position == -1)
    {
        priv->text_x = actor_width - text_width;
        priv->cursor_pos.x += priv->text_x;
    }
    else
    {
        if (priv->text_x <= 0)
        {
            gint diff = -1 * priv->text_x;

            if (cursor_x < diff)
                priv->text_x += diff - cursor_x;
            else if (cursor_x > (diff + actor_width))
                priv->text_x -= cursor_x - (diff+actor_width);
        }

        priv->cursor_pos.x += priv->text_x;
    }
}
else
{
    priv->text_x = 0;
}

```

```

priv->fgcol.alpha = clutter_actor_get_opacity (self);
pango_clutter_render_layout (priv->layout,
                             priv->text_x, 0,
                             &priv->fgcol, 0);

if (EXAMPLE_MULTILINE_ENTRY_GET_CLASS (entry)->paint_cursor)
    EXAMPLE_MULTILINE_ENTRY_GET_CLASS (entry)->paint_cursor (entry);
}

static void
example_multiline_entry_request_coords (ClutterActor      *self,
                                       ClutterActorBox *box)
{
    ExampleMultilineEntry *entry = EXAMPLE_MULTILINE_ENTRY (self);
    ExampleMultilineEntryPrivate *priv = entry->priv;
    gint width;

    width = CLUTTER_UNITS_TO_DEVICE (box->x2 - box->x1);

    if (priv->width != width)
    {
        example_multiline_entry_clear_layout (entry);
        example_multiline_entry_ensure_layout (entry, width);

        priv->width = width;
    }

    CLUTTER_ACTOR_CLASS (example_multiline_entry_parent_class)->request_coords (self, box);
}

static void
example_multiline_entry_dispose (GObject *object)
{
    ExampleMultilineEntry      *self = EXAMPLE_MULTILINE_ENTRY (object);
    ExampleMultilineEntryPrivate *priv;

    priv = self->priv;

    if (priv->layout)
    {
        g_object_unref (priv->layout);
        priv->layout = NULL;
    }

    if (priv->context)
    {
        g_object_unref (priv->context);
        priv->context = NULL;
    }

    G_OBJECT_CLASS (example_multiline_entry_parent_class)->dispose (object);
}

```

```

static void
example_multiline_entry_finalize (GObject *object)
{
    ExampleMultilineEntryPrivate *priv = EXAMPLE_MULTILINE_ENTRY (object)->priv;

    if (priv->desc)
        pango_font_description_free (priv->desc);

    g_free (priv->text);
    g_free (priv->font_name);

    G_OBJECT_CLASS (example_multiline_entry_parent_class)->finalize (object);
}

static void
example_multiline_entry_class_init (ExampleMultilineEntryClass *klass)
{
    GObjectClass      *gobject_class = G_OBJECT_CLASS (klass);
    ClutterActorClass *actor_class = CLUTTER_ACTOR_CLASS (klass);

    klass->paint_cursor = example_multiline_entry_paint_cursor;

    actor_class->paint          = example_multiline_entry_paint;
    actor_class->request_coords = example_multiline_entry_request_coords;

    gobject_class->finalize      = example_multiline_entry_finalize;
    gobject_class->dispose       = example_multiline_entry_dispose;
    gobject_class->set_property  = example_multiline_entry_set_property;
    gobject_class->get_property  = example_multiline_entry_get_property;

    /**
     * ExampleMultilineEntry:font-name:
     *
     * The font to be used by the entry, expressed in a string that
     * can be parsed by pango_font_description_from_string().
     */
    g_object_class_install_property
        (gobject_class, PROP_FONT_NAME,
         g_param_spec_string ("font-name",
                              "Font Name",
                              "Pango font description",
                              NULL,
                              G_PARAM_READABLE | G_PARAM_WRITABLE));
    /**
     * ExampleMultilineEntry:text:
     *
     * The text inside the entry.
     */
    g_object_class_install_property
        (gobject_class, PROP_TEXT,
         g_param_spec_string ("text",

```

```

"Text",
"Text to render",
NULL,
G_PARAM_READABLE | G_PARAM_WRITABLE));
/**
 * ExampleMultilineEntry:color:
 *
 * The color of the text inside the entry.
 *
 */
g_object_class_install_property
  (gobject_class, PROP_COLOR,
   g_param_spec_boxed ("color",
"Font Colour",
"Font Colour",
CLUTTER_TYPE_COLOR,
G_PARAM_READABLE | G_PARAM_WRITABLE));

/**
 * ExampleMultilineEntry::text-changed:
 * @entry: the actor which received the event
 *
 * The ::text-changed signal is emitted after @entry's text changes
 */
entry_signals[TEXT_CHANGED] =
  g_signal_new ("text-changed",
               G_TYPE_FROM_CLASS (gobject_class),
               G_SIGNAL_RUN_LAST,
               G_STRUCT_OFFSET (ExampleMultilineEntryClass, text_changed),
               NULL, NULL,
               g_cclosure_marshal_VOID__VOID,
               G_TYPE_NONE, 0);

/**
 * ExampleMultilineEntry::cursor-event:
 * @entry: the actor which received the event
 * @geometry: a #ClutterGeometry
 *
 * The ::cursor-event signal is emitted each time the input cursor's geometry
 * changes, this could be a positional or size change. If you would like to
 * implement your own input cursor, set the cursor-visible property to %FALSE,
 * and connect to this signal to position and size your own cursor.
 *
 */
entry_signals[CURSOR_EVENT] =
  g_signal_new ("cursor-event",
               G_TYPE_FROM_CLASS (gobject_class),
               G_SIGNAL_RUN_LAST,
               G_STRUCT_OFFSET (ExampleMultilineEntryClass, cursor_event),
               NULL, NULL,
               g_cclosure_marshal_VOID__BOXED,
               G_TYPE_NONE, 1,

```

```

CLUTTER_TYPE_GEOMETRY | G_SIGNAL_TYPE_STATIC_SCOPE);

g_type_class_add_private (gobject_class, sizeof (ExampleMultilineEntryPrivate));
}

static void
example_multiline_entry_init (ExampleMultilineEntry *self)
{
    ExampleMultilineEntryPrivate *priv;
    gdouble resolution;
    gint font_size;

    self->priv = priv = EXAMPLE_MULTILINE_ENTRY_GET_PRIVATE (self);

    resolution = clutter_backend_get_resolution (clutter_get_default_backend ());
    if (resolution < 0)
        resolution = 96.0; /* fall back */

    if (G_UNLIKELY (_context == NULL))
    {
        _font_map = PANGO_CLUTTER_FONT_MAP (pango_clutter_font_map_new ());
        pango_clutter_font_map_set_resolution (_font_map, resolution);
        _context = pango_clutter_font_map_create_context (_font_map);
    }

    priv->layout          = NULL;
    priv->text            = NULL;
    priv->position        = -1;
    priv->text_x          = 0;

    priv->fgcol.red       = 0;
    priv->fgcol.green     = 0;
    priv->fgcol.blue      = 0;
    priv->fgcol.alpha    = 255;

    priv->font_name       = g_strdup (DEFAULT_FONT_NAME);
    priv->desc            = pango_font_description_from_string (priv->font_name);

    /* We use the font size to set the default width and height, in case
     * the user doesn't call clutter_actor_set_size().
     */
    font_size = PANGO_PIXELS (pango_font_description_get_size (priv->desc)
                              * resolution
                              / 72.0);
    clutter_actor_set_size (CLUTTER_ACTOR (self), font_size * 20, 50);

    priv->cursor          = clutter_rectangle_new_with_color (&priv->fgcol);
    clutter_actor_set_parent (priv->cursor, CLUTTER_ACTOR (self));
}

/**
 * example_multiline_entry_new:

```

```

*
* Creates a new, empty #ExampleMultilineEntry.
*
* Returns: the newly created #ExampleMultilineEntry
*/
ClutterActor *
example_multiline_entry_new (void)
{
    ClutterActor *entry = g_object_new (CLUTTER_TYPE_MULTILINE_ENTRY,
                                        NULL);

    clutter_actor_set_size (entry, 50, 50);

    return entry;
}

/**
 * example_multiline_entry_get_text:
 * @entry: a #ExampleMultilineEntry
 *
 * Retrieves the text displayed by @entry.
 *
 * Return value: the text of the entry. The returned string is
 * owned by #ExampleMultilineEntry and should not be modified or freed.
 */
G_CONST_RETURN gchar *
example_multiline_entry_get_text (ExampleMultilineEntry *entry)
{
    g_return_val_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry), NULL);

    return entry->priv->text;
}

/**
 * example_multiline_entry_set_text:
 * @entry: a #ExampleMultilineEntry
 * @text: the text to be displayed
 *
 * Sets @text as the text to be displayed by @entry. The
 * ExampleMultilineEntry::text-changed signal is emitted.
 */
void
example_multiline_entry_set_text (ExampleMultilineEntry *entry,
                                 const gchar *text)
{
    ExampleMultilineEntryPrivate *priv;

    g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));
    g_return_if_fail (text != NULL);

    priv = entry->priv;

    g_object_ref (entry);

```

```

g_free (priv->text);

priv->text = g_strdup (text);
priv->n_chars = g_utf8_strlen (priv->text, -1);

example_multiline_entry_clear_layout (entry);
example_multiline_entry_clear_cursor_position (entry);

if (CLUTTER_ACTOR_IS_VISIBLE (entry))
    clutter_actor_queue_redraw (CLUTTER_ACTOR (entry));

g_signal_emit (G_OBJECT (entry), entry_signals[TEXT_CHANGED], 0);

g_object_notify (G_OBJECT (entry), "text");
g_object_unref (entry);
}

/**
 * example_multiline_entry_get_font_name:
 * @entry: a #ExampleMultilineEntry
 *
 * Retrieves the font used by @entry.
 *
 * Return value: a string containing the font name, in a format
 * understandable by pango_font_description_from_string(). The
 * string is owned by #ExampleMultilineEntry and should not be modified
 * or freed.
 */
G_CONST_RETURN gchar *
example_multiline_entry_get_font_name (ExampleMultilineEntry *entry)
{
    g_return_val_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry), NULL);

    return entry->priv->font_name;
}

/**
 * example_multiline_entry_set_font_name:
 * @entry: a #ExampleMultilineEntry
 * @font_name: a font name and size, or %NULL for the default font
 *
 * Sets @font_name as the font used by @entry.
 *
 * @font_name must be a string containing the font name and its
 * size, similarly to what you would feed to the
 * pango_font_description_from_string() function.
 */
void
example_multiline_entry_set_font_name (ExampleMultilineEntry *entry,
                                       const gchar *font_name)
{
    ExampleMultilineEntryPrivate *priv;

```

```

PangoFontDescription *desc;

g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));

if (!font_name || font_name[0] == '\0')
    font_name = DEFAULT_FONT_NAME;

priv = entry->priv;

if (strcmp (priv->font_name, font_name) == 0)
    return;

desc = pango_font_description_from_string (font_name);
if (!desc)
{
    g_warning ("Attempting to create a PangoFontDescription for "
"font name '%s', but failed.",
font_name);
    return;
}

g_object_ref (entry);

g_free (priv->font_name);
priv->font_name = g_strdup (font_name);

if (priv->desc)
    pango_font_description_free (priv->desc);

priv->desc = desc;

if (entry->priv->text && entry->priv->text[0] != '\0')
{
    example_multiline_entry_clear_layout (entry);

    if (CLUTTER_ACTOR_IS_VISIBLE (entry))
clutter_actor_queue_redraw (CLUTTER_ACTOR (entry));
}

g_object_notify (G_OBJECT (entry), "font-name");
g_object_unref (entry);
}

/**
 * example_multiline_entry_set_color:
 * @entry: a #ExampleMultilineEntry
 * @color: a #ClutterColor
 *
 * Sets the color of @entry.
 */
void
example_multiline_entry_set_color (ExampleMultilineEntry *entry,

```

```

        const ClutterColor *color)
{
    ClutterActor *actor;
    ExampleMultilineEntryPrivate *priv;

    g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));
    g_return_if_fail (color != NULL);

    priv = entry->priv;

    g_object_ref (entry);

    priv->fgcol.red = color->red;
    priv->fgcol.green = color->green;
    priv->fgcol.blue = color->blue;
    priv->fgcol.alpha = color->alpha;

    actor = CLUTTER_ACTOR (entry);

    clutter_actor_set_opacity (actor, priv->fgcol.alpha);

    clutter_rectangle_set_color (CLUTTER_RECTANGLE (priv->cursor), &priv->fgcol);

    if (CLUTTER_ACTOR_IS_VISIBLE (actor))
        clutter_actor_queue_redraw (actor);

    g_object_notify (G_OBJECT (entry), "color");
    g_object_unref (entry);
}

/**
 * example_multiline_entry_get_color:
 * @entry: a #ExampleMultilineEntry
 * @color: return location for a #ClutterColor
 *
 * Retrieves the color of @entry.
 */
void
example_multiline_entry_get_color (ExampleMultilineEntry *entry,
    ClutterColor *color)
{
    ExampleMultilineEntryPrivate *priv;

    g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));
    g_return_if_fail (color != NULL);

    priv = entry->priv;

    color->red = priv->fgcol.red;
    color->green = priv->fgcol.green;
    color->blue = priv->fgcol.blue;
    color->alpha = priv->fgcol.alpha;
}

```

```

/**
 * example_multiline_entry_set_cursor_position:
 * @entry: a #ExampleMultilineEntry
 * @position: the position of the cursor.
 *
 * Sets the position of the cursor. The @position must be less than or
 * equal to the number of characters in the entry. A value of -1 indicates
 * that the position should be set after the last character in the entry.
 * Note that this position is in characters, not in bytes.
 */
static void
example_multiline_entry_set_cursor_position (ExampleMultilineEntry *entry,
                                             gint                    position)
{
    ExampleMultilineEntryPrivate *priv;
    gint len;

    g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));

    priv = entry->priv;

    if (priv->text == NULL)
        return;

    len = g_utf8_strlen (priv->text, -1);

    if (position < 0 || position >= len)
        priv->position = -1;
    else
        priv->position = position;

    example_multiline_entry_clear_cursor_position (entry);

    if (CLUTTER_ACTOR_IS_VISIBLE (entry))
        clutter_actor_queue_redraw (CLUTTER_ACTOR (entry));
}

/**
 * example_multiline_entry_handle_key_event:
 * @entry: a #ExampleMultilineEntry
 * @kev: a #ClutterKeyEvent
 *
 * This function will handle a #ClutterKeyEvent, like those returned in a
 * key-press/release-event, and will translate it for the @entry. This includes
 * non-alphanumeric keys, such as the arrows keys, which will move the
 * input cursor. You should use this function inside a handler for the
 * ClutterStage::key-press-event or ClutterStage::key-release-event.
 */
void
example_multiline_entry_handle_key_event (ExampleMultilineEntry *entry,
                                           ClutterKeyEvent *kev)

```

```

{
  ExampleMultilineEntryPrivate *priv;
  gint pos = 0;
  gint len = 0;
  gint keyval = clutter_key_event_symbol (kev);

  g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));

  priv = entry->priv;

  pos = priv->position;

  if (priv->text)
    len = g_utf8_strlen (priv->text, -1);

  switch (keyval)
  {
    case CLUTTER_Escape:
    case CLUTTER_Shift_L:
    case CLUTTER_Shift_R:
      /* Ignore these - Don't try to insert them as characters. */
      break;

    case CLUTTER_BackSpace:
      /* Delete the current character: */
      if (pos != 0 && len != 0)
        example_multiline_entry_delete_chars (entry, 1);
      break;

    case CLUTTER_Delete:
    case CLUTTER_KP_Delete:
      /* Delete the current character: */
      if (len && pos != -1)
        example_multiline_entry_delete_text (entry, pos, pos+1);
      break;

    case CLUTTER_Left:
    case CLUTTER_KP_Left:
      /* Move the cursor one character left: */
      if (pos != 0 && len != 0)
        {
          if (pos == -1)
            example_multiline_entry_set_cursor_position (entry, len - 1);
          else
            example_multiline_entry_set_cursor_position (entry, pos - 1);
        }
      break;

    case CLUTTER_Right:
    case CLUTTER_KP_Right:
      /* Move the cursor one character right: */
      if (pos != -1 && len != 0)
        {

```

```

        if (pos != len)
            example_multiline_entry_set_cursor_position (entry, pos + 1);
    }
    break;

case CLUTTER_Up:
case CLUTTER_KP_Up:
    /* TODO: Calculate the index of the position on the line above,
       and set the cursor to it. */
    break;

case CLUTTER_Down:
case CLUTTER_KP_Down:
    /* TODO: Calculate the index of the position on the line below,
       and set the cursor to it. */
    break;

case CLUTTER_End:
case CLUTTER_KP_End:
    /* Move the cursor to the end of the text: */
    example_multiline_entry_set_cursor_position (entry, -1);
    break;

case CLUTTER_Begin:
case CLUTTER_Home:
case CLUTTER_KP_Home:
    /* Move the cursor to the start of the text: */
    example_multiline_entry_set_cursor_position (entry, 0);
    break;

default:
    example_multiline_entry_insert_unichar (entry,
                                           clutter_keysym_to_unicode (keyval));
    break;
}
}

/**
 * example_multiline_entry_insert_unichar:
 * @entry: a #ExampleMultilineEntry
 * @wc: a Unicode character
 *
 * Insert a character to the right of the current position of the cursor,
 * and updates the position of the cursor.
 *
 */
static void
example_multiline_entry_insert_unichar (ExampleMultilineEntry *entry,
                                       gunichar wc)
{
    ExampleMultilineEntryPrivate *priv;
    GString *new = NULL;

```

```

glong pos;

g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));
g_return_if_fail (g_unichar_validate (wc));

if (wc == 0)
    return;

priv = entry->priv;

g_object_ref (entry);

new = g_string_new (priv->text);
pos = offset_to_bytes (priv->text, priv->position);
new = g_string_insert_unichar (new, pos, wc);

example_multiline_entry_set_text (entry, new->str);

if (priv->position >= 0)
    example_multiline_entry_set_cursor_position (entry, priv->position + 1);

g_string_free (new, TRUE);

g_object_notify (G_OBJECT (entry), "text");
g_object_unref (entry);
}

/**
 * example_multiline_entry_delete_chars:
 * @entry: a #ExampleMultilineEntry
 * @len: the number of characters to remove.
 *
 * Characters are removed from before the current position of the cursor.
 */
static void
example_multiline_entry_delete_chars (ExampleMultilineEntry *entry,
                                     guint                    num)
{
    ExampleMultilineEntryPrivate *priv;
    GString *new = NULL;
    gint len;
    gint pos;
    gint num_pos;

    g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));

    priv = entry->priv;

    if (!priv->text)
        return;

    g_object_ref (entry);

```

```

len = g_utf8_strlen (priv->text, -1);
new = g_string_new (priv->text);

if (priv->position == -1)
{
    num_pos = offset_to_bytes (priv->text, len - num);
    new = g_string_erase (new, num_pos, -1);
}
else
{
    pos = offset_to_bytes (priv->text, priv->position - num);
    num_pos = offset_to_bytes (priv->text, priv->position);
    new = g_string_erase (new, pos, num_pos-pos);
}
example_multiline_entry_set_text (entry, new->str);

if (priv->position > 0)
    example_multiline_entry_set_cursor_position (entry, priv->position - num);

g_string_free (new, TRUE);

g_object_notify (G_OBJECT (entry), "text");
g_object_unref (entry);
}

/**
 * example_multiline_entry_delete_text:
 * @entry: a #ExampleMultilineEntry
 * @start_pos: the starting position.
 * @end_pos: the end position.
 *
 * Deletes a sequence of characters. The characters that are deleted are
 * those characters at positions from @start_pos up to, but not including,
 * @end_pos. If @end_pos is negative, then the characters deleted will be
 * those characters from @start_pos to the end of the text.
 */
void
example_multiline_entry_delete_text (ExampleMultilineEntry      *entry,
                                     gssize                       start_pos,
                                     gssize                       end_pos)
{
    ExampleMultilineEntryPrivate *priv;
    GString *new = NULL;
    gint start_bytes;
    gint end_bytes;

    g_return_if_fail (EXAMPLE_IS_MULTILINE_ENTRY (entry));

    priv = entry->priv;

    if (!priv->text)

```

```
    return;

    start_bytes = offset_to_bytes (priv->text, start_pos);
    end_bytes = offset_to_bytes (priv->text, end_pos);

    new = g_string_new (priv->text);
    new = g_string_erase (new, start_bytes, end_bytes - start_bytes);

    example_multiline_entry_set_text (entry, new->str);

    g_string_free (new, TRUE);
}
```

# Chapter 9. Contributing

If you find errors in this documentation or if you would like to contribute additional material, you are encouraged to write an email to [clutter@o-hand.com](mailto:clutter@o-hand.com). Thanks.