The Enlightened Widget Library (EWL) book

dan 'dj2' sinclair <zero@everburning.com>

The Enlightened Widget Library (EWL) book

by dan 'dj2' sinclair

Abstract

A developers guide to the Enlightened Widget Library (EWL). This book attempts to show how to use the EWL, from simply creating a window to writing a full video player. Along with introducing the EWL the book aims to serve as a complete reference to the library including code examples for each of the different widgets.

Table of Contents

List of Figures

List of Examples

Chapter 1. Introduction

The EWL is a library for creating graphical user interfaces based upon the Enlightenment Foundation Libraries (EFL). The library aims to ease the pain of creating graphical interfaces. With that in mind the library aims for simplicity and elegance.

The EWL is authored primarily by:

- Nathan 'RbdPngn' Ingersoll

The EWL works in a similar fashion to other widget libraries, in that it is based on a callback system. As elements are created and added to the interface any desired event callbacks are registered and these functions will be triggered when the specified events happen. This simple callback method makes it possible to build very complex interfaces from a relatively simple set of base primitives.

This tutorial is an attempt to familiarize the user with the different aspects of the EWL. The tutorial will probably never completely document all aspects of EWL as the system continues to grow. A good understanding of C programming is assumed throughout the tutorial.

If you have any troubles with either this tutoral, or using the EWL in general any feedback is greatly appreciated as it would help improve either the tutorial or the EWL itself. Please see the Contributing section for more information.

Chapter 2. Getting Started

Getting EWL installed

Before using EWL you need to have the libraries installed on your computer. EWL can be retrieved from the Enlightenment CVS and directions on how this is done can be found at: `http://www.enlightenment.org/pages/source.html` [http://www.enlightenment.org/pages/source.html] along with detailed installation directions.

You will need to install a lot of dependencies before being able to install EWL, this is because it depends on many of the EFL libraries being present on the system.

Once you have the other EFL libraries installed, installing EWL is as simple as:

```
./configure;
make;
sudo make install;
```

Creating a simple Window

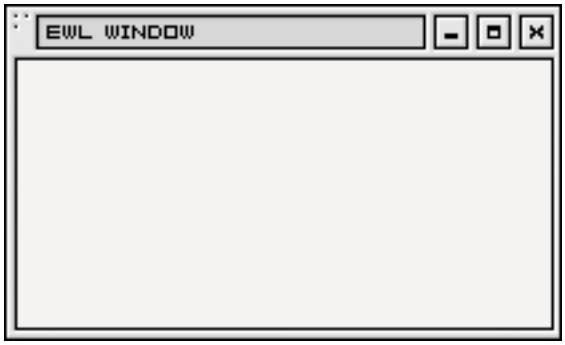
The first step in creating an EWL application is to get the main window to be displayed on the screen.

```
#include <stdio.h>
#include <Ewl.h>

void destroy_cb(Ewl_Widget *w, void *event, void *data) {
    ewl_widget_destroy(w);
    ewl_main_quit();
}

int main(int argc, char ** argv) {
    Ewl_Widget *win = NULL;

    if (!ewl_init(&argc, argv)) {
        printf("Unable to init ewl\n");
        return 1;
    }

    win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(win), "EWL Window");
    ewl_window_name_set(EWL_WINDOW(win), "EWL_WINDOW");
    ewl_window_class_set(EWL_WINDOW(win), "EWLWindow");
    ewl_object_size_request(EWL_OBJECT(win), 200, 100);
    ewl_callback_append(win, EWL_CALLBACK_DELETE_WINDOW, destroy_cb, NULL);
    ewl_widget_show(win);

    ewl_main();
    return 0;
}
```

This program can be compiled with a simple:

```
zero@oberon [create_window] -> gcc -o create_window main.c \
 `ewl-config --cflags --libs`
```

And if executed should produce something similar to:



Now that we know what we're making, lets go over the code in more detail.

```
#include <stdio.h>
#include <Ewl.h>
```

All EWL applications will start with the <Ewl.h> include. This will pull in all of the other header files that EWL requires to function.

```
void destroy_cb(Ewl_Widget *w, void *event, void *data) {
    ewl_widget_destroy(w);
    ewl_main_quit();
}
```

The `destroy_cb` will be used by EWL when the window manager requests the application terminate. Callbacks will be described further in the Callbacks section.

The `ewl_widget_destroy()` is used to signal to EWL that we no longer need the given widget, in this case the window, and for EWL to clean up the resources used by that widget.

Finally, we call `ewl_main_quit()` which causes EWL to exit its main processing loop and return from the `ewl_main()` function.

```
int main(int argc, char ** argv) {
    Ewl_Widget *win = NULL;

    if (!ewl_init(&argc, argv)) {
        printf("Unable to init ewl\n");
        return 1;
    }
```

Before we can actually use EWL we must initialize the library. This is done through the call to `ewl_init()`. We pass the argc and argv parameters from main to EWL as there are a few specific switches EWL parses from the arguments.

These switches currently include:

EWL command line switches

- --ewl-theme <name>
- --ewl-print-theme-keys
- --ewl-segv
- --ewl-software-x11
- --ewl-gl-x11
- --ewl-fb

The <name> parameter to the --ewl-theme switch is the name of the theme you wish to be used. This can be either located in one of the system directories, or in the local directory.

If EWL was able to successfully initialize itself the call to `ewl_init()` will return a value > 0. If it was unsuccessful there is no real point in continuing as EWL will not function correctly.

```
    win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(win), "EWL Window");
    ewl_window_name_set(EWL_WINDOW(win), "EWL_WINDOW");
    ewl_window_class_set(EWL_WINDOW(win), "EWLWindow");
    ewl_object_size_request(EWL_OBJECT(win), 200, 100);
    ewl_callback_append(win, EWL_CALLBACK_DELETE_WINDOW, destroy_cb, NULL);
    ewl_widget_show(win);
```

This is where the actual window is created. A call to `ewl_window_new()` creates the new, empty window. We then take that window and start attaching data. We begin by setting the title with `ewl_window_title_set()`, which will set the string to be displayed by the window manager on the top of the window. The next two function calls, `ewl_window_name_set()` and `ewl_window_class_set()` set data that will be used by the window manager to better manage your application.

We then proceed to set the base size for the window with a call to `ewl_object_size_request()`. The second and third parameters (200, 100) specify the width and height we wish the window to have on creation. You'll notice that this call casts to `EWL_OBJECT()`. This is because of the hierarchy of widgets that EWL provides, (further described in the Object Hierarchy chapter) an ewl_window is a ewl_object so we can use the ewl_object operations on an ewl_window.

We then proceed to add the delete callback to the window with a call to `ewl_callback_append`. The second parameter of which is the type of signal we wish to listen too, the third is the function to call and finally the fourth is any user data to be sent to the callback.

Once the window is all set up and ready to go, a simple call to `ewl_widget_show()` will have EWL display the window.

```
    ewl_main();
    return 0;
}
```

The call to `ewl_main()` will tell EWL to start its main processing loop waiting on any signals. `ewl_main()` will handle the shutdown of EWL when the main processing loop is exited.

That's it. Although it's probably one of the simplest EWL applications that can be produced, it will be used as a basis for many of the other examples presented in this tutorial, and many EWL applications that are produced.

Hello World

Once you have a window on the screen its time to do something more fun with it. So, following in the

grand tradition, something with Hello in it.

I am only going to explain the portions of the program which have not already been seen. If there is something you do not understand please reference the previous section and it should be explained there.

```
#include <stdio.h>
#include <Ewl.h>

void destroy_cb(Ewl_Widget *w, void *event, void *data) {
    ewl_widget_destroy(w);
    ewl_main_quit();
}

void text_update_cb(Ewl_Widget *w, void *event, void *data) {
    char *s = NULL;
    Ewl_Widget *label = NULL;
    char buf[BUFSIZ];

    s = ewl_text_text_get(EWL_TEXT(w));
    label = (Ewl_Widget *)data;

    snprintf(buf, BUFSIZ, "Hello %s", s);
    ewl_text_text_set(EWL_TEXT(label), buf);

    free(s);
    return;
}

int main(int argc, char ** argv) {
    Ewl_Widget *win = NULL;
    Ewl_Widget *box = NULL;
    Ewl_Widget *label = NULL;
    Ewl_Widget *o = NULL;

    /* init the library */
    if (!ewl_init(&argc, argv)) {
        printf("Unable to initialize EWL\n");
        return 1;
    }

    /* create the window */
    win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(win), "Hello world");
    ewl_window_class_set(EWL_WINDOW(win), "hello");
    ewl_window_name_set(EWL_WINDOW(win), "hello");
    ewl_object_size_request(EWL_OBJECT(win), 200, 50);
    ewl_callback_append(win, EWL_CALLBACK_DELETE_WINDOW, destroy_cb, NULL);
    ewl_widget_show(win);

    /* create the container */
    box = ewl_vbox_new();
    ewl_container_child_append(EWL_CONTAINER(win), box);
    ewl_object_fill_policy_set(EWL_OBJECT(box), EWL_FLAG_FILL_ALL);
    ewl_widget_show(box);

    /* create text label */
    label = ewl_text_new();
    ewl_container_child_append(EWL_CONTAINER(box), label);
    ewl_object_alignment_set(EWL_OBJECT(label), EWL_FLAG_ALIGN_CENTER);
    ewl_text_styles_set(EWL_TEXT(label), EWL_TEXT_STYLE_SOFT_SHADOW);
    ewl_text_color_set(EWL_TEXT(label), 255, 0, 0, 255);
    ewl_text_text_set(EWL_TEXT(label), "hello");
    ewl_widget_show(label);
```

```
    /* create the entry */
    o = ewl_entry_new();
    ewl_container_child_append(EWL_CONTAINER(box), o);
    ewl_object_alignment_set(EWL_OBJECT(o), EWL_FLAG_ALIGN_CENTER);
    ewl_object_padding_set(EWL_OBJECT(o), 5, 5, 5, 0);
    ewl_text_color_set(EWL_TEXT(o), 0, 0, 0, 255);
    ewl_callback_append(o, EWL_CALLBACK_VALUE_CHANGED, text_update_cb, label);
    ewl_widget_show(o);

    ewl_main();
    return 0;
}
```

If you compile and run this application, in the same fashion as the first example, you should see something similar to the following window.

This one's a bit longer than the simple creating of a window, but then it also includes more functionality. If you run this program you should see a simple window with a bit of text saying 'Hello' at the top and a text area. Typing in the text area and hitting 'Enter' will display 'Hello' plus whatever you've typed.

The 'Hello' string has had a bit of text styling applied. You can see that the text has had a simple colour change applied and is displayed with a drop shadow.

Now that you know what it does, lets take a look at the new bits of code this example introduce.

```
void text_update_cb(Ewl_Widget *w, void *event, void *data) {
    char *s = NULL;
    Ewl_Widget *label = NULL;
    char buf[BUFSIZ];

    s = ewl_text_text_get(EWL_TEXT(w));
    label = (Ewl_Widget *)data;

    snprintf(buf, BUFSIZ, "Hello %s", s);
    ewl_text_text_set(EWL_TEXT(label), buf);

    free(s);
    return;
}
```

The `text_update_cb()` is the callback we are going to register for when the user has pressed 'Enter' in the text field. It has the same signature as the destroy callback, and all other EWL callbacks that we will be registering.

The text that has been entered is retrieved with a call to `ewl_text_text_get()` passing the entry widget from which we want to retrieve the text. This will return a pointer to the text string, it is the applications responsibility to free this pointer.

We then cast the `data` parameter into a `Ewl_Widget`. This is because, as you will see in the register

callback, we are attaching a widget to this callback as a data parameter.

We can then take this new text and replace the contents of the current text label by calling `ewl_text_text_set()` passing the text object and the text to be displayed.

```
box = ewl_vbox_new();
ewl_container_child_append(EWL_CONTAINER(win), box);
ewl_object_fill_policy_set(EWL_OBJECT(box), EWL_FLAG_FILL_ALL);
ewl_widget_show(box);
```

While we could just attach any widgets onto the main application window, it is a bit cleaner to attach the widgets into a box that is attached to the main window. The box is created with a call to `ewl_vbox_new()` creating a vertical box layout. We could have used `ewl_hbox_new()` if we desired a horizontal box instead of a vertical one. Once the box is created we attach it to the window by calling `ewl_container_child_append()`. This places the given widget into the container as the next element. Containers are packed from top to bottom, or left to right, so the order elements are inserted into the container effect there appearance on screen. Lastly, before showing the widget, we attach a fill policy with `ewl_object_fill_policy_set()`. The fill policy changes the way the object fills in its available space.

The possible fill policies are:

EWL Fill Flags

- EWL_FLAG_FILL_NONE
- EWL_FLAG_FILL_HSHRINK
- EWL_FLAG_FILL_VSHRINK
- EWL_FLAG_FILL_SHRINK
- EWL_FLAG_FILL_HFILL
- EWL_FLAG_FILL_VFILL
- EWL_FLAG_FILL_FILL
- EWL_FLAG_FILL_ALL

All of which should be pretty self explanatory, with the exceptions of, `EWL_FLAG_FILL_SHRINK`, `EWL_FLAG_FILL_FILL` and `EWL_FLAG_FILL_ALL`. The SHRINK flag is the or of the two HSHRINK and VSHRINK flags. The FILL flag is the or of the two HFILL and VFILL flags. Finally the ALL flag is the or of the two SHRINK and FILL flags.

```
label = ewl_text_new();
ewl_container_child_append(EWL_CONTAINER(box), label);
ewl_object_alignment_set(EWL_OBJECT(label), EWL_FLAG_ALIGN_CENTER);
ewl_text_styles_set(EWL_TEXT(label), EWL_TEXT_STYLE_SOFT_SHADOW);
ewl_text_color_set(EWL_TEXT(label), 255, 0, 0, 255);
ewl_text_text_set(EWL_TEXT(label), "Hello");
ewl_widget_show(label);
```

Now that we have our containing box set up, we create the actual text element that is going to function as our label. The label is created with a call to `ewl_text_new()` in this case, we pass NULL as the value because we will be specifying our text after we attach some styling to the object. You can also pass a text string into `ewl_text_new()` if desired. Just keep in mind that text styling happens for text that is added *after* the styling is attached.

Once the widget is created we attach it to the box with `ewl_container_child_append()`. Next we set the alignment on the text object though `ewl_object_alignment_set()`. This specifies how the contents will be aligned within the widget itself.

The alignment function will accept one of:

EWL Alignment Flags

- EWL_FLAG_ALIGN_CENTER
- EWL_FLAG_ALIGN_LEFT
- EWL_FLAG_ALIGN_RIGHT
- EWL_FLAG_ALIGN_TOP
- EWL_FLAG_ALIGN_BOTTOM

Once all the widget properties are specified, we attach some text formatting properties to the widget. The first, `ewl_text_style_set()`, sets the style of the text object. The styles change the appearance of the text by applying some kind of filter, in this case, creating a 'soft shadow' appearance to the widget. We then set the colour of the text to red by calling `ewl_text_color_set()`. There are four parameters to the colour function, those being, red, green, blue and alpha.

```
o = ewl_entry_new();
ewl_container_child_append(EWL_CONTAINER(box), o);
ewl_object_alignment_set(EWL_OBJECT(o), EWL_FLAG_ALIGN_CENTER);
ewl_object_padding_set(EWL_OBJECT(o), 5, 5, 5, 0);
ewl_text_color_set(EWL_ENTRY(o), 0, 0, 0, 255);
ewl_callback_append(o, EWL_CALLBACK_VALUE_CHANGED, text_update_cb, label);
ewl_widget_show(o);
```

The final widget we create is a text entry box. This is done with a call to `ewl_entry_new()`. In this case we are giving "" as the value, but an initial string could be given to be displayed in the entry box. We do a similar set of initializations to the entry box, setting the alignment and setting the text colour to black. The call to `ewl_object_padding_set()` sets the amount of padding around the widget. The four parameters are, left, right, top and bottom.

With that you should have a basic understanding of how EWL functions and how different widgets are created and setup.

Callbacks

The EWL is powered through the use of callbacks. A large amount of the internal work of the library itself also works on callbacks.

A callback is a function that will be called when a specific event happens. These events can be anything from the user clicking a button to the window being destroyed by the window manager.

For the events that an application needs notification a callback is registered through EWL. This is done with the `ewl_callback_append()`. This function takes four parameters: the object to attach the callback too, the callback desired, the callback function and any user data.

Some of the possible callbacks include:

Possible EWL Callbacks

EWL_CALLBACK_DESTROY	The widget is freed
EWL_CALLBACK_DELETE_WINDOW	The window is being closed

EWL_CALLBACK_KEY_DOWN	A key was pressed down
EWL_CALLBACK_KEY_UP	A key was released
EWL_CALLBACK_MOUSE_DOWN	Mouse button was pressed down
EWL_CALLBACK_MOUSE_UP	Mouse button was released
EWL_CALLBACK_MOUSE_MOVE	Mouse was moved
EWL_CALLBACK_MOUSE_WHEEL	Mouse wheel scrolled
EWL_CALLBACK_FOCUS_IN	Mouse was placed over the widget
EWL_CALLBACK_FOCUS_OUT	Mouse was moved away from the widget
EWL_CALLBACK_SELECT	Widget was selected by mouse or key
EWL_CALLBACK_DESELECT	Widget was deselected by mouse or key
EWL_CALLBACK_CLICKED	Mouse was pressed and released on a widget
EWL_CALLBACK_DOUBLE_CLICKED	Mouse was clicked twice quickly
EWL_CALLBACK_HILITED	Mouse is over the widget
EWL_CALLBACK_VALUE_CHANGED	Value in widget changed

The callback function has a signature like `void fcn(Ewl_Widget *, void *, void *)`. The first parameter is the widget that activated this callback. The second parameter is the event data and the third parameter is the user attached data.

The event data is a type that relates to the callback itself. So, for example, when the callback for `EWL_CALLBACK_MOUSE_WHEEL` is called the event data will have a struct of type `Ewl_Event_Mouse_Wheel` and this struct contains additional information about the event. In the wheel case, the key modifiers, the mouse position and the direction of scroll.

The last parameter to the callback attach function is the user data. This allows you to attach any data desired to be passed to the callback when it is executed. This data will be provided to the callback in the form of its third parameter.

Chapter 3. Object Hierarchy

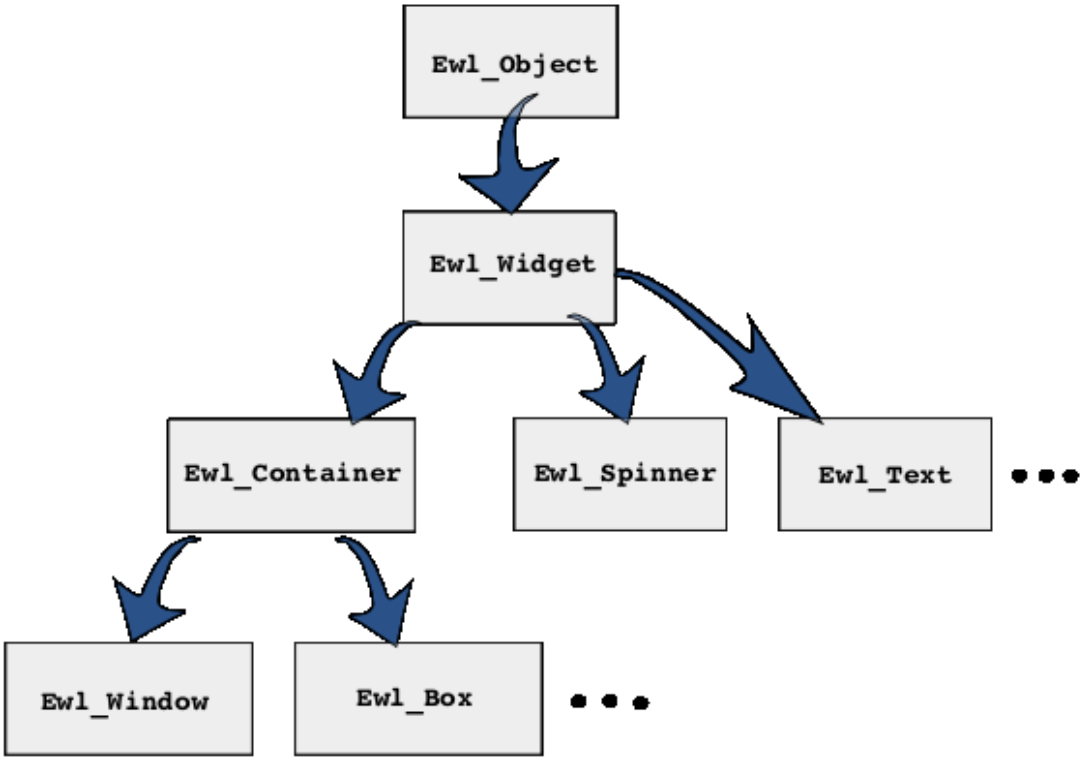
Introduction

The EWL widgets are setup in a hierarcy. The base widget that everything extends from is the `Ewl_Object`. The `Ewl_Object` provides all of the base functionality for each widget including the sizing, alignment, fill policies, padding and others. This is the main building block of the EWL. An application using EWL will never need to allocate an `Ewl_Object`

Sitting just above the `Ewl_Object` is the `Ewl_Widget`. Again, all widgets inheret from this object, which in turn inherits from the `Ewl_Object`. This object provides the base functinality for a widget to interact with users. Like the `Ewl_Object` an application will never need to allocate an `Ewl_Widget` itself.

With the `Ewl_Widget` in place we can start to build up the hierarchy of widgets that form the EWL. The hierarchy looks something similar to that in the EWL Object Hierarcy figure below.

Figure 3.1. The EWL Object Hierarchy



The `Ewl_Container` object is built off of the `Ewl_Widget` object and provides the functionality for widgets that are to hold other widgets. This includes anything from the main window, to boxes, to scrollpanes.

Object Casting

As you progress into EWL you will notice that there is a lot of casting between different types. To make this easier, each cast to a particular type has a EWL_TYPE() macro defined. So for example there are EWL_OBJECT(o) and EWL_WIDGET(o) defined to make life easier.

These macros should always be used when converting between EWL widgets so that you know that the right thing is being done.

Adding new widgets

To add new widgets into EWL you just need to create a new struct that has the appropriate type of sub-class as the first element. This subclass object must not be a pointer.

Example 3.1. Creating EWL Widgets

```
struct Ewl_Foo {
    Ewl_Container container;
    int bar;
}
```

This would create a new Ewl_Foo widget that inherits from the `Ewl_Container` so you would be able to pack other widgets into this new widget type.

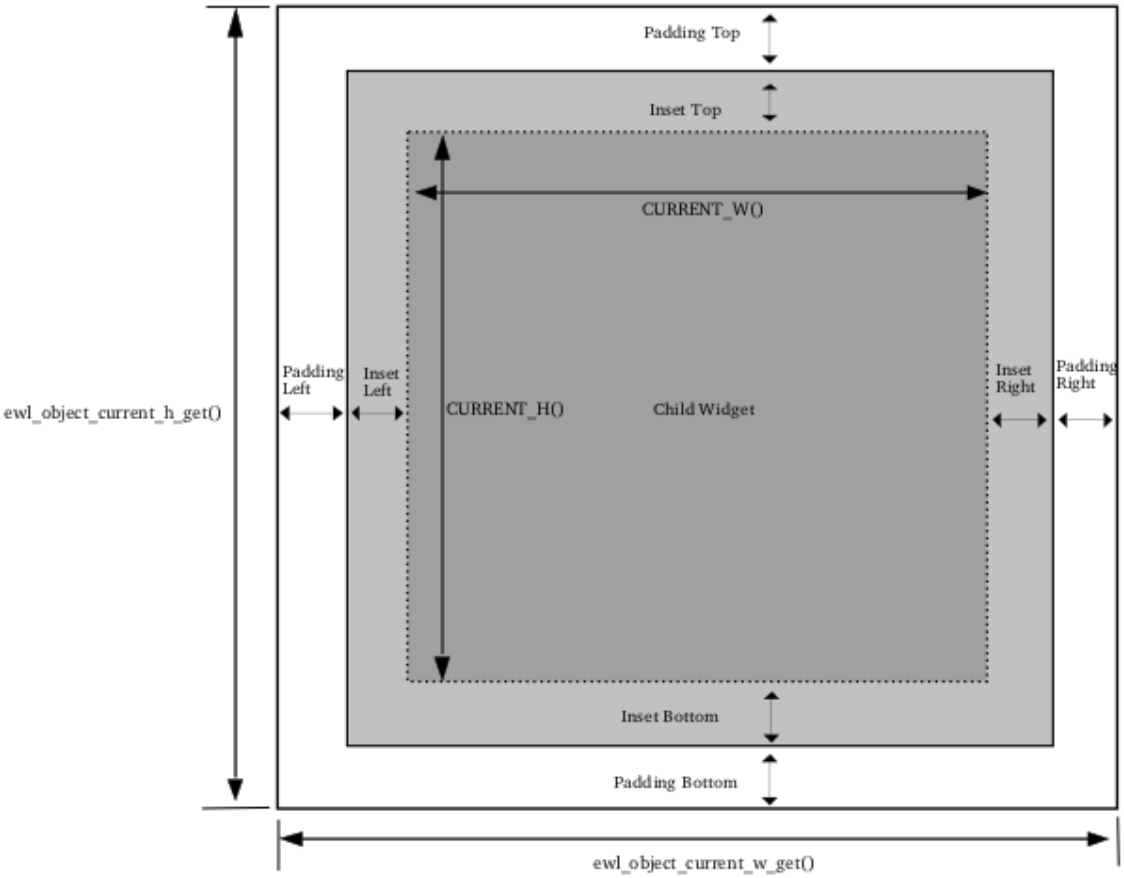
Chapter 4. Widget Packing

As you're writing an EWL application you will need to start laying out the widgets into the different boxes. To do so, you'll need a bit of information on how EWL packs widgets together.

There are two main orientations for containers in EWL, vertical or horizontal. The different orientations can be most easily seen in the section called “Ewl_Box”.

In EWL, each widget has an amount of padding around the widget and an inset set into the widget. This is seen in the below in Figure 4.1, “Padding and Insets”.

Figure 4.1. Padding and Insets



Chapter 5. Configuration

EWL configuration

EWL uses the Ecore_Config system to handle all of its configuration data. This makes the changing of values easy though the existing tools to work with Ecore_Config.

The following are the current keys used by EWL with a brief description.

/ewl/debug/enable	Enable debug mode
/ewl/debug/level	Set the debug level [0 - 10]
/ewl/evas/render_method	Set the method Evas will use to render the display. This can be one of: • software_x11 • gl_x11 • fb For X11 software, X11 OpenGL and Framebuffer display respectively.
/ewl/evas/font_cache	The size of the Evas font cache
/ewl/evas/image_cache	The size of the Evas image cache
/ewl/theme/name	The name of the EWL theme to use (minus the .eet portion)
/ewl/theme/cache	A boolean to indicate if theme values should be cached by EWL
/ewl/theme/print_keys	a boolean to indicate if the theme key names should be printed as they are accessed
/ewl/theme/color_classes/override	Override the default colour classes
/ewl/theme/color_classes/count	The number of colour classes that are overridden
/ewl/theme/color_class/[n]/name	The name of the [n]th colour class
/ewl/theme/color_class/[n][rgba]	A key for each of the r, g, b, a values of the colour class

Application configuration

The best way for an application to handle its specific configuration is to also use Ecore_Config. Doing so is simple and already handles things like defaults and callbacks for data changes.

As a safety precaution you should probably make a call to `ecore_init()` in your code before using the Ecore_Config functions. This will guarantee that Ecore won't be shutdown before you're finished using it. This means you will need to make a call to `ecore_shutdown()` when you're finished using Ecore_Config.

Before you start using Ecore_Config you must make a call to `int ecore_config_init(char *)` where the parameter is the name you want your configuration to appear under in Ecore_Config. This is also the name that would be used with **examine** to change your configuration data. When you are finished using Ecore_Config you should call `int ecore_config_shutdown(void)` to close down

the Ecore_Config system.

Chapter 6. EWL Themes

As the EWL was designed around the lower EFL libraries it incorporates the use of Edje to handle the widget theming. You can either identify the theme to be used in the main configuration db, **examine _system**, or specified on the command line with the `--ewl-theme` flag.

In order to theme the EWL, a background in Edje is required. You can take a look at the EdjeBook for an overview and reference manual to EDC programming.

As you add widgets into your EWL application, EWL builds up an appearance tree. This can be seen using the `--ewl-print-theme-keys` flag when running any EWL application. This flag will print out each of the appearance keys as EWL attempts to access them. You will see something similar to Example 6.1, “EWL theme keys”.

Example 6.1. EWL theme keys

```
zero@oberon [e17] -> ewl_test --ewl-print-theme-keys
/theme/font_path
/window/file
/window/group
/window/hbox/file
/window/hbox/group
/window/hbox/tree/file
/window/hbox/tree/group
/window/hbox/scrollpane/file
/window/hbox/scrollpane/group
/window/hbox/tree/row/file
/window/hbox/tree/row/group
/window/hbox/tree/scrollpane/file
/window/hbox/tree/scrollpane/group
```

As EWL attempts to locate these entries in the theme db it will remove successive portions of the path until it finds the key in the theme. So, in the case of `/window/hbox/scrollpane/group` it will traverse through:

- `/window/hbox/scrollpane/group`
- `/hbox/scrollpane/group`
- `/scrollpane/group`

attempting to locate a matching key.

Using this hierarichal structure EWL allows for very specific theme keys to be set. It will allow you to theme your boxes differently if they appear inside a scrollpane or inside of a window. This delivers a lot of power and flexibility into your hands as a themer.

The easiest way to figure out how to theme EWL is to take a look at the current themes and how they're written. If you look into the *data/themes* directory you will see several directories and several .edc files. Each .edc and directory combination is a theme. This top level .edc file is what pulls all of the pieces of the lower directories together.

This chapter will be pulling its examples from the 'zero' theme that currently resides in the EWL cvs tree.

Taking a look at the top level zero.edc file, you can see a large `data {}` section. This contains the mappings that EWL uses from the appearance keys metioned above to the Edje groups defined in the

theme files. This mapping is done with the keys that end in */group*.

You'll notice looking at the .edc file that the `/bar/group` key points to the `bar` group. So, somewhere in the other .edc files there is a Edje group called `bar`. So using this method you could then theme a bar that appears inside a scrollpane by using the full path from the scrollpane down to the bar element in the data section.

If you want different fonts for you different widgets this is also done in the `data {}` section of the main .edc file. A key that ends in `/text/font` will set the font on that text item. Similarly there is a `/text/font_size` and `/text/style` for setting the font size and style respectively.

As an example the `item, "/button/text/font" "vio";` will set the text used on any button widgets to use the "vio" alias which was defined in the `fonts` section with: `font: "zero/fonts/vio.ttf" "vio";`

There are several different keys that can be looked up by EWL as the program executes. Its best to use the `--ewl-print-theme-keys` option to find the ones you desire.

The main `data {}` section also includes a little bit of meta-data about the theme itself. There are currently four relevant keys:

- /theme/author
- /theme/font_path
- /theme/license
- /theme/name

Chapter 7. Widgets

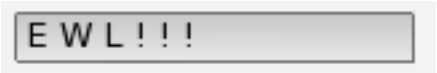
We will now look at each widget individually. See the code that creates the widget and a screen shot of the widget in action (if applicable).

Button widgets

Ewl_Button

The button widget is simply a widget with a label attached. When the user clicks on the button the callback attached to `EWL_CALLBACK_CLICKED` will be executed.

Figure 7.1. An Ewl Button



Example 7.1. Creating a button

```
    Ewl_Widget *button = ewl_button_new("A button");
    ewl_object_alignment_set(EWL_OBJECT(button), EWL_FLAG_ALIGN_CENTER);
    ewl_callback_append(button, EWL_CALLBACK_CLICKED, button_cb, NULL);
    ewl_widget_show(button);
```

The label portion of the button can be aligned to any of the `EWL_FLAG_ALIGN_*` settings.

Example 7.2. Button Callback

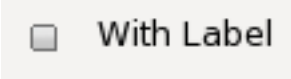
```
void button_cb(Ewl_Widget *w, void *event, void *data) {
    printf("button pressed\n");
}
```

The label on a button can be manipulated after the button has been created through the two calls:

- `char *ewl_button_label_get(EwlButton *)`
- `void ewl_button_label_set(EwlButton *, char *)`

Ewl_Checkbutton

Figure 7.2. An Ewl Checkbutton



Example 7.3. Creating a checkbutton

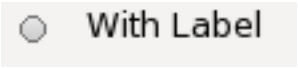
```
    Ewl_Widget *cb = ewl_checkbutton_new("Label");
    ewl_checkbutton_label_position_set(EWL_CHECKBUTTON(cb), EWL_FLAG_ALIGN_LEFT);
    ewl_callback_append(cb, EWL_CALLBACK_VALUE_CHANGED, checkbutton_cb, NULL);
    ewl_widget_show(cb);
```

Example 7.4. Button Callback

```
void checkbutton_cb(Ewl_Widget *w, void *event, void *data) {
    if (ewl_checkbutton_is_checked(EWL_CHECKBUTTON(w)))
        printf("checked\n");
    else
        printf("Not checked\n");
}
```

Ewl_Radiobutton

Figure 7.3. An EWL radiobutton



Container widgets

Ewl_Box

The box widgets allow you to specify different ways in which the application will be laid out. You can create either a horizontal (hbox) or vertical (vbox) box. A vertical box will have its children packed from top to bottom while a horizontal box will have its widgets packed from left to right.

A box widget will not show up in the application itself, it is just used as a container for other widgets.

Example 7.5. Creating EWL boxes

```
Ewl_Widget *hbox = ewl_hbox_new();
ewl_widget_show(hbox);

Ewl_Widget *vbox = ewl_vbox_new();
ewl_widget_show(vbox);
```

The box widgets are relativly simple to create and use, only requiring a call to the new function.

The functions to manipulate the boxes include:

- `void ewl_box_orientation_set(Ewl_Box *, Ewl_Orientation)`
- `Ewl_Orientation ewl_box_orientation_get(Ewl_Box *)`
- `void ewl_box_spacing_set(Ewl_Box *, int)`
- `void ewl_box_homogeneous_set(Ewl_Box *, int)`

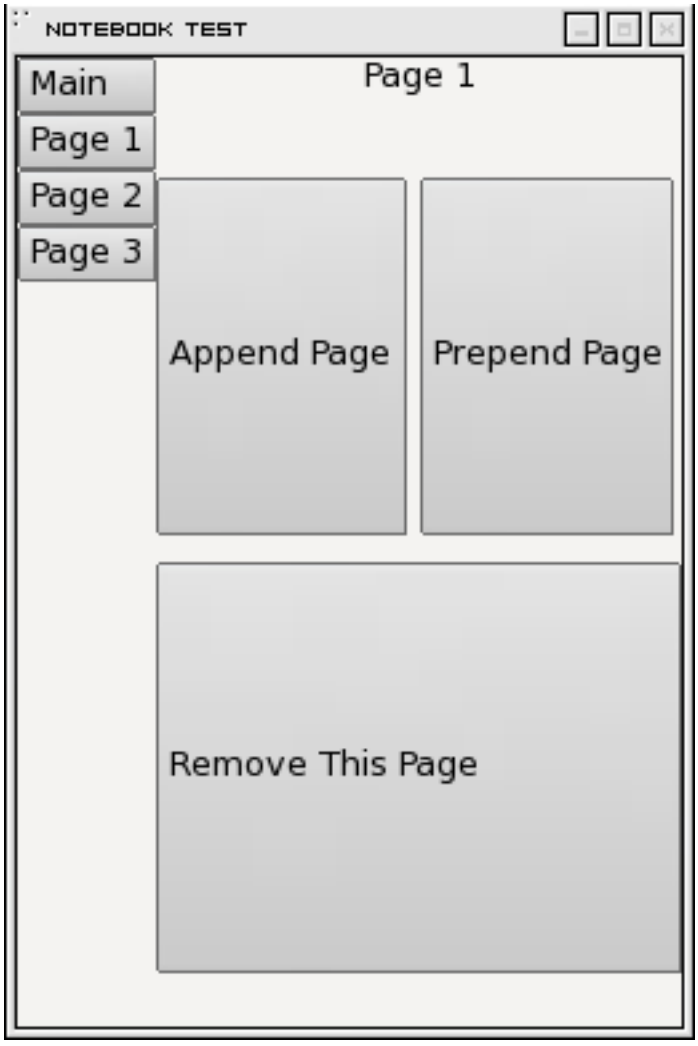
The `Ewl_Orientation` flag can be one of:

- EWL_ORIENTATION_HORIZONTAL
- EWL_ORIENTATION_VERTICAL

The `ewl_box_spacing_set()` will set the amount of spacing between the items in the box to the given value. While the `ewl_box_homogeneous_set()` will set the box to give all items in it the same size if this is set to true, otherwise they will have their required size.

Ewl_Notebook

Figure 7.4. An EWL Notebook



Ewl_Scrollpane

Ewl_Table

Ewl_Tree

The tree widget allows for laying out widgets in a series of expandable rows. When creating a tree, the number of columns can be specified, and future changes will allow for changing the number and location of columns at runtime.

Example 7.6. Creating EWL trees

```
Ewl_Widget *tree = ewl_tree_new(number_columns);
```

```
ewl_widget_show(tree);
```

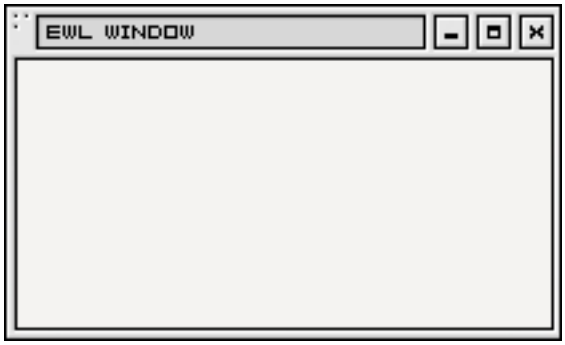
The functions to manipulate the tree include:

- `void ewl_tree_headers_set(Ewl_Tree *, char **)`
- `void ewl_tree_columns_set(Ewl_Tree *, unsigned short)`
- `Ecore_List *ewl_tree_selected_get(Ewl_Tree *)`
- `void ewl_tree_selected_clear(Ewl_Tree *)`
- `Ewl_Tree_Mode ewl_tree_mode_get(Ewl_Tree *)`
- `void ewl_tree_mode_set(Ewl_Tree *, Ewl_Tree_Mode)`
- `void ewl_tree_row_add(Ewl_Tree *, Ewl_Row *, Ewl_Widget **)`
- `void ewl_tree_text_row_add(Ewl_Tree *, Ewl_Row *, char **)`
- `void ewl_tree_entry_row_add(Ewl_Tree *, Ewl_Row *, char **)`
- `void ewl_tree_row_destroy(Ewl_Tree *, Ewl_Row *)`
- `void ewl_tree_row_expand_set(Ewl_Row *, Ewl_Tree_Node_Flags)`
- `Ewl_Row *ewl_tree_row_find(Ewl_Row *, Ewl_Tree_Node_Flags)`

Ewl_Window

An ewl_window will be used by every EWL application. This is the window that will display all of the other desired EWL widgets.

Figure 7.5. An EWL Window



Example 7.7. Creating a Window

```
Ewl_Widget *window = ewl_window_new();
ewl_window_title_set(EWL_WINDOW(window), "foo window");
ewl_window_class_set(EWL_WINDOW(window), "foo_class");
ewl_window_name_set(EWL_WINDOW(window), "foo_name");
ewl_object_size_request(EWL_OBJECT(window), 300, 400);
ewl_callback_append(window, EWL_CALLBACK_DELETE_WINDOW, win_del_cb, NULL);
ewl_widget_show(window);
```

Setting up the basic window is pretty simple. We take the extra steps of calling: `ewl_window_title_set()`, `ewl_window_name_set()` and `ewl_window_class_set()` to fill in the information the window manager uses.

Since the window is a `Ewl_Object` like any other, we use the `ewl_object_size_request()` to request the starting size of our window. We could have also called `ewl_object_minimum_size_set()` and `ewl_object_maximum_size_set()` to constrain the minimum/maximum sizes of our window.

The main callback used by a Ewl_Window is the `EWL_CALLBACK_DELETE_WINDOW`. This will be called when the window is being destroyed by the window manager. It should be used to cleanup any resources that the application has used before exiting the application.

Example 7.8. Ewl Window destroy callback

```
void win_del_cb(Ewl_Widget *w, void *event, void *data) {
    ewl_widget_destroy(w);
    ewl_main_quit();
}
```

Some of the other operations involving the Ewl_Window object are:

- `char *ewl_window_title_get(Ewl_Window *)`
- `char *ewl_window_name_get(Ewl_Window *)`
- `char *ewl_window_class_get(Ewl_Window *)`
- `void ewl_window_borderless_set(Ewl_Window *)`
- `void ewl_window_move(Ewl_Window *, int x, int y)`
- `void ewl_window_position_get(Ewl_Window *, int *x, int *y)`

The first three calls are pretty self explanatory. The `ewl_window_borderless_set()` can be used to tell the window manager not to display any decoration around the window, this includes the border and the title bar. The function `ewl_window_move()` is used to position the window to a specific place on the desktop, indexed from the top left corner. There is also a `ewl_window_position_get()` which will return the position of the window on the desktop.

Range widgets

Ewl_Progressbar

Figure 7.6. An EWL progress bar

Ewl_Seeker

Figure 7.7. An EWL seeker

Example 7.9. Creating an EWL seeker

```
    Ewl_Widget *s = ewl_seeker_new(EWL_ORIENTATION_HORIZONTAL);
    ewl_range_value_set(EWL_RANGE(s), 5.0);
    ewl_range_range_set(EWL_RANGE(s), 10.0);
    ewl_range_step_set(EWL_RANGE(s), 1);
    ewl_callback_append(s, EWL_CALLBACK_VALUE_CHANGED, seeker_cb, NULL);
    ewl_widget_show(s);
```

Example 7.10. Ewl_Seeker callback

```
void seeker_cb(Ewl_Widget *w, void *event, void *data) {
    double val = ewl_range_value_get(EWL_RANGE(w));
    printf("%f\n", val);
}
```

Ewl_Spinner

Figure 7.8. An EWL spinner

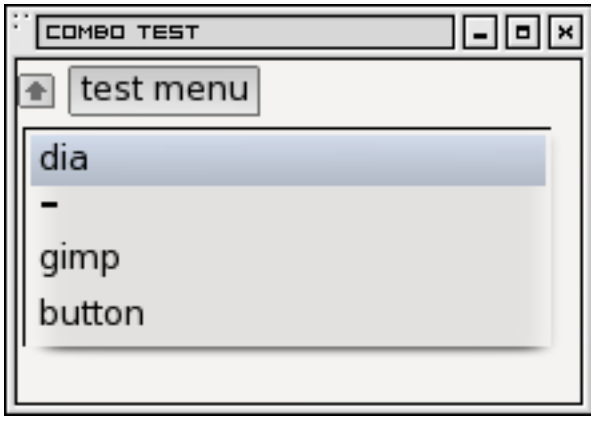
Menu widgets

Ewl_Menu

Other widgets

Ewl_Combo

Figure 7.9. An Ewl Combo box



Example 7.11. Creating a combo box

```
    Ewl_Widget *combo = ewl_combo_new("combo box");
    ewl_callback_append(combo, EWL_CALLBACK_VALUE_CHANGED,
                                combo_change_cb, NULL);
    ewl_widget_show(combo);

    Ewl_Widget *item1 = ewl_menu_item_new(NULL, "foo");
    ewl_container_child_append(EWL_CONTAINER(combo), item1);
    ewl_widget_show(item1);
```

Example 7.12. combo box value changed callback

```
void combo_change_cb(Ewl_Widget *w, void *event, void *data) {
    char *text = (char *)event;
    printf("Value changed to %s\n", text);
}
```

Ewl_Dialog

The `Ewl_Dialog` widget provides a way to display a simple dialog box to the user which can then prompt for a response, give warnings or just display simple messages.

Figure 7.10. An Ewl Dialog

Example 7.13. EWL Dialog code

```
Ewl_Widget *dialog = NULL;
Ewl_Widget *o = NULL;

o = ewl_text_new();
ewl_text_text_set(EWL_TEXT(o), "a dialog eh");
ewl_object_alignment_set(EWL_OBJECT(o),
EWL_FLAG_ALIGN_CENTER);
ewl_widget_show(o);

dialog = ewl_dialog_new(EWL_POSITION_BOTTOM);
ewl_dialog_has_separator_set(EWL_DIALOG(dialog), 0);
ewl_dialog_widget_add(EWL_DIALOG(dialog), o);
ewl_object_alignment_set(EWL_OBJECT(dialog), EWL_FLAG_ALIGN_CENTER);
ewl_widget_show(dialog);

o = ewl_dialog_button_add(EWL_DIALOG(dialog),"OK", EWL_RESPONSE_OK);
ewl_container_child_append(EWL_CONTAINER(dialog), o);
ewl_callback_append(o, EWL_CALLBACK_CLICKED, dialog_clicked_cb, dialog);
ewl_widget_show(o);

o = ewl_dialog_button_left_add(EWL_DIALOG(dialog),"Cancel", EWL_RESPONSE_CANCE
ewl_container_child_append(EWL_CONTAINER(dialog), o);
ewl_callback_append(o, EWL_CALLBACK_CLICKED, dialog_clicked_cb, dialog);
ewl_widget_show(o);
```

This example will create an `Ewl_Dialog` with two buttons: an OK button and a Cancel button. The dialog itself is created with the call to `ewl_dialog_new()` passing the position of the buttons relative to the window itself. The possible values are:

- EWL_POSITION_TOP
- EWL_POSITION_BOTTOM
- EWL_POSITION_LEFT
- EWL_POSITION_RIGHT

A `Ewl_Dialog` can optionally have a horizontal line drawn to seperate the two sections of the dialog. The line is controlled with the `ewl_dialog_has_separator_set()` where 0 means do not draw separator and 1 means to draw the separator. There is a corresponding `ewl_dialog_has_separator_get()` returning 1 if there is a separator and 0 otherwise.

The content of the main display area of the box is controlled through the function `ewl_dialog_widget_add()`. In this instance we add a `Ewl_Text` object into the dialog.

Once the dialog is initialized we need to create the desired buttons. The buttons are created by calling `ewl_dialog_button_add()` for the 'OK' button and `ewl_dialog_button_left_add()` for the 'Cancel' button. The parameters are the dialog, the label of the button and the response code to return from the button. There are several pre-defined labels available, including:

- EWL_STOCK_OK
- EWL_STOCK_APPLY
- EWL_STOCK_CANCEL
- EWL_STOCK_OPEN
- EWL_STOCK_SAVE
- EWL_STOCK_PAUSE
- EWL_STOCK_PLAY
- EWL_STOCK_STOP

The pre-defined response codes are:

- EWL_RESPONSE_OPEN
- EWL_RESPONSE_SAVE
- EWL_RESPONSE_OK
- EWL_RESPONSE_CANCEL
- EWL_RESPONSE_APPLY
- EWL_RESPONSE_PLAY
- EWL_RESPONSE_PAUSE
- EWL_RESPONSE_STOP

Once the buttons are created they need to be added to the dialog and have a callback append for there `EWL_CALLBACK_CLICKED` state.

Example 7.14. EWL Dialog callback

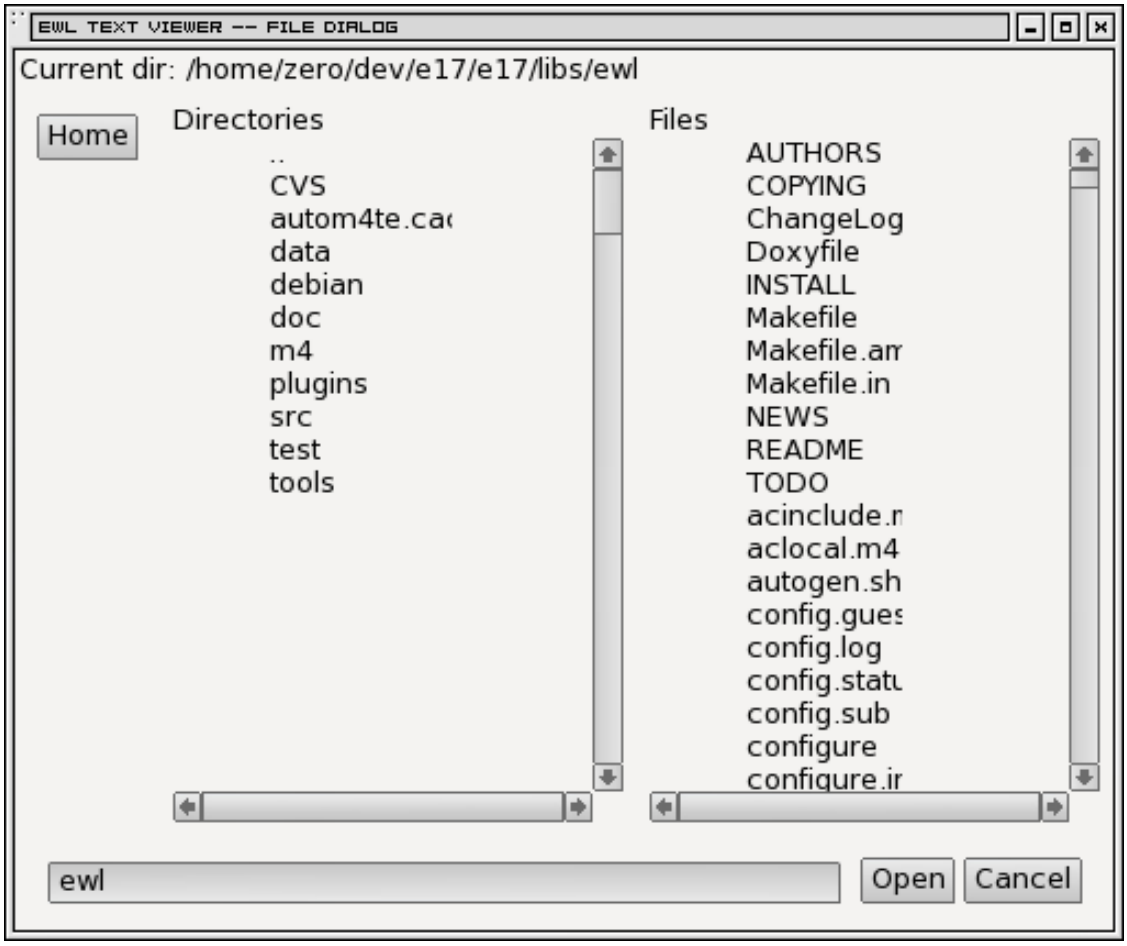
```
void dialog_clicked_cb(Ewl_Widget *w, void *event, void *data) {
    int d = EWL_BUTTON_STOCK(w)->response_id;

    if (d == EWL_RESPONSE_OK)
        printf("OK\n");
    else if (d == EWL_RESPONSE_CANCEL)
        printf("CANCEL\n");

    ewl_widget_destroy(EWL_WIDGET(data));
}
```

The response code of the button that was clicked is available from the `Ewl_Button_Stock` widget itself through its response_id parameter. Using this value we can determine which of the buttons was clicked. We also passed the `Ewl_Dialog` itself through the data parameter so that we could destroy the dialog when we were finished.

Ewl_Filedialog

It is often desired to allow the user to open and save files. This can be easily accomplished through the use of the `Ewl_Filedialog`.

Figure 7.11. An EWL file dialog



This file dialog has been embedded into its own window, but it could have been placed in another window in the same fashion.

Example 7.15. Creating an EWL filedialog

```
Ewl_Widget *filedialog = ewl_filedialog_new(EWL_FILEDIALOG_TYPE_OPEN);
ewl_callback_append(filedialog, EWL_CALLBACK_VALUE_CHANGED,
                            open_file_cb, NULL);
ewl_widget_show(filedialog);
```

When the file dialog is created you specify a type either `EWL_FILDIALOG_TYPE_OPEN` or `EWL_FILEDIALOG_TYPE_SAVE` depending on the type of file dialog desired. The callback `EWL_CALLBACK_VALUE_CHANGED` will be executed when the user clicks the 'Open' button in the dialog.

It is also possible to pack other widgets into the filedialog itself. This is done through the normal `ewl_container_child_append()`. So, if you needed, for example, to add a 'Home' button, you could create the button and pack it into the file dialog where it will appear down the left side.

You can change the directory that is currently being viewed in the file dialog by executing `void ewl_filedialog_set_directory(Ewl_Filedialog *, char *path)` where `path` is

the full path to the desired directory.

Example 7.16. Ewl_Filedialog open callback

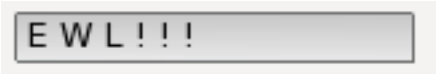
```
void open_file_cb(Ewl_Widget *w, void *event, void *data) {
    char *filename = (char *)event;
    printf("selected file %s\n", filename);
}
```

The file that has been selected is passed to the callback as the `event` parameter. If you wish to remove the filedialog you can do something similar to `ewl_widget_hide(fd_win)` where `fd_win` is the window object holding the file dialog.

Ewl_Entry

The EWL entry box is available when you need to retrieve text input from the user. The box works on single lines, and the callback is triggered when the user presses the 'Enter' key.

Figure 7.12. An EWL entry box



Example 7.17. Creating an EWL entry box

```
   Ewl_Widget *entry = ewl_entry_new();
   ewl_object_size_request(EWL_OBJECT(entry), 100, 15);
   ewl_object_padding_set(EWL_OBJECT(entry), 1, 1, 1, 1);
   ewl_callback_append(entry, EWL_CALLBACK_VALUE_CHANGED, entry_cb, NULL);
   ewl_widget_show(entry);
```

The `Ewl_Entry` is a fairly simple object to work with, about the only required setup is to create the new object and attach a callback for `EWL_CALLBACK_VALUE_CHANGED` events. This example takes the extra steps of setting the size with `ewl_object_size_request()` and adding a little bit of padding to the widget with `ewl_object_padding_set()`.

Example 7.18. Ewl_Entry value changed callback

```
void entry_cb(Ewl_Widget *w, void *event, void *data) {
    char *s = ewl_entry_text_get(EWL_ENTRY(w));
    printf("%s\n", s);
```

```
    ewl_entry_text_set(EWL_ENTRY(w), "New Text");
}
```

This callback grabs the current value of the entry widget with the call to `ewl_entry_text_get()` and then resets the text to the value of 'New Text' by calling `ewl_entry_text_set()`.

The `Ewl_Entry` object allows you to set whether or not the text is editable with a call to `void ewl_entry_editable_set(Ewl_Entry *, unsigned int edit)` where `edit` is 0 for uneditable and editable otherwise.

Ewl_Password

The `Ewl_Password` widget provides similar functionality to the `Ewl_Text` widget, except that any text entered will not be displayed, instead a configurable obscuring character will be displayed.

Figure 7.13. An EWL password dialog

Example 7.19. Creating an EWL password

```
    Ewl_Widget *p = ewl_password_new("default");
    ewl_password_obscure_set(EWL_PASSWORD(p), "-");
    ewl_callback_append(p, EWL_CALLBACK_VALUE_CHANGED, passwd_cb, NULL);
    ewl_widget_show(p);
```

The default obscuring character used is a '*' character. This can be easily changed by calling `ewl_password_obscure_set(Ewl_Password *, char)`. There is also a corresponding `char ewl_password_obscure_get(Ewl_Password *)` to retrieve the current obscuring character. As with the `ewl_text` widget there are two functions to get and set the text of the widget: `ewl_password_text_set(Ewl_Password *, char *)` and `char *ewl_password_text_get(Ewl_Password *)`.

When the user presses the enter key in the password box a `EWL_CALLBACK_VALUE_CHANGED` will be triggered.

Example 7.20. Ewl_Password value changed callback

```
void passwd_cb(Ewl_Widget *, void *event, void *data) {
    char *text = ewl_password_text_get(EWL_PASSWORD(w));
    printf("text: %s\n", text);
}
```

Ewl_Image

Example 7.21. Ewl_Image

```
Ewl_Widget *i = ewl_image_new();
ewl_image_file_set(EWL_IMAGE(i), "/usr/foo/img.png", NULL);
ewl_widget_show(i);
```

The `ewl_image_new()` function takes two parameters, the path to the image to be loaded and a key for the image data. The key is used primarily to load edje groups or keyed data as the image.

Ewl_Text

The `Ewl_Text` widget provides for a multi-line text layout widget. It can be utillized whenever the display of text is required in an application. It works well with the `Ewl_Scrollpane` to provide a scrollable text area.

Example 7.22. Ewl_Text code

```
Ewl_Widget *text = ewl_text_new();
ewl_text_text_set(EWL_TEXT(text), "text");
ewl_widget_show(text);
```

Creating the basic `Ewl_Text` object is pretty simple, the object will be setup to diplay the parameter to `ewl_text_new()`.

Once the text object is created you can change the text, retrieve the current text contents or get the text length with:

- `ewl_text_text_set(Ewl_Text *, char *)`
- `ewl_text_text_prepend(Ewl_Text *, char *)`
- `ewl_text_text_append(Ewl_Text *, char *)`
- `ewl_text_text_insert(Ewl_Text *, char *, int index)`
- `char *ewl_text_text_get(Ewl_Text *)`
- `int ewl_text_length_get(Ewl_Text *)`

The `Ewl_Text` widget allows you to preform styling changes to the text in the widget. Different portions of the text can be different colours, fonts or styles. The styling that is applied to a widget is based on what is setup when the text is added to the widget. So, if you want your text to be red, you need to set the colour of the `Ewl_Text` object *before* adding the text.

The colour of the text can be manipulated with the `ewl_text_color_set(Ewl_Text *, int r, int g, int b, int a)` call while the current colour information can be retrieved with the `ewl_text_color_get(Ewl_Text *, int *r, int *g, int *b, int *a)`.

The font settings of the text can be manipulated with the `ewl_text_font_set(Ewl_Text *, char *font, int size)` call. With the calls to get the current font name as size defined as: `char *ewl_text_font_get(Ewl_Text *)` and `int ewl_text_font_size_get(Ewl_Text`

*).

To retrieve or set the alignment of the text widget there are the two functions: `ewl_text_align_set(Ewl_Text *, unsigned int align)` and `unsigned int ewl_text_align_get(Ewl_Text *)`. Where the align parameter is one of the EWL alignment flags:

- EWL_FLAG_ALIGN_CENTER
- EWL_FLAG_ALIGN_LEFT
- EWL_FLAG_ALIGN_RIGHT
- EWL_FLAG_ALIGN_TOP
- EWL_FLAG_ALIGN_BOTTOM

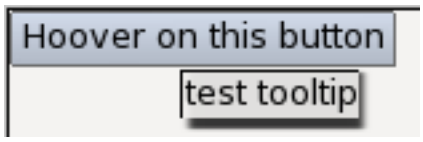
It is also possible to set the style of the text. This can include things such as bolding the text or setting soft shadows. The styles that are available are shipped through the Etox library and currently include:

- bold
- outline
- plain
- raised
- shadow
- soft_shadow

Ewl_Tooltip

The `Ewl_Tooltip` widget provides a simple popup dislpaying information about the widgets in your application. The tooltip appears after the mouse has hovered over the widget of a period of time.

Figure 7.14. An EWL tooltip



Example 7.23. Ewl Tooltip code

```
Ewl_Widget *t = ewl_tooltip_new(parent);
ewl_toolip_text_set(t, "this is the tooltip");
```

The `ewl_tooltip_new()` function takes as its parameter, the Ewl_Widget that the tooltip is to be associated with. A tooltip may contain multiple lines of text.

An Ewl_Tooltip should be appended to the parent Ewl_Window of the widget it is associated with.

Unlike most other widgets, you should not use the ewl_widget_show() function wit h the Ewl_Tooltip.

Once the tooltip is created, you can modify the text that the tooltip displays along with changing the delay before the tooltip appears. These modifications are done with:

- `ewl_tooltip_text_set(Ewl_Tooltip *t, char *txt)`
- `ewl_tooltip_delay_set(Ewl_Tooltip *t, double delay)`

Ewl_Media

The `Ewl_Media` widget allows for the embedding of video objects into your application. This is done by wrapping around the Emotion library.

Figure 7.15. An EWL media object

Example 7.24. Ewl_Media code

```
Ewl_Media *m = ewl_media_new(file);
ewl_callback_append(m, EWL_CALLBACK_REALIZE, video_realize_cb, NULL);
ewl_callback_append(m, EWL_CALLBACK_VALUE_CHANGED, video_change_cb, NULL);
ewl_widget_show(m);
```

Creating the basic video object is no harder then creating the object and showing it (assuming you've appended it to whatever container it is being placed into). We hook the two callbacks `EWL_CALLBACK_REALIZE` and `EWL_CALLBACK_VALUE_CHANGED`. We hook in the realize callback so we can determine the length of the video to be displayed if desired. This is only available after the video has been realized, and will return 0 until it has been realized. The value change callback will be called whenever emotion advances the video. This can be used to setup a timer, or a seek bar and have it auto advance for the video.

Example 7.25. Ewl_Media callbacks

```
void video_realize_cb(Ewl_Widget *w, void *event, void *data) {
    double len = ewl_media_length_get(EWL_MEDIA(video));
}

void video_change_cb(Ewl_Widget *w, void *event, void *data) {
    char buf[512];
    int h, m;
    double s;

    ewl_media_position_time_get(EWL_MEDIA(video), &h, &m, &s);
    snprintf(buf, sizeof(buf), "%02i:%02i:%02.0f", h, m, s);
}
```

The video that is being displayed can be changed by calling `ewl_media_media_set(Ewl_Media *, char *)` or if you just wish to know what is currently playing you can call `char *ewl_media_media_get(Ewl_Media *)`. The length of the current video can be retrieved by calling `int ewl_media_length_get(Ewl_Media *)`. The length can also be retrieved as a time value by calling `ewl_media_length_time_get(Ewl_Media *, int h, int m, double s)`.

You can start the video playing by passing 1 to `ewl_media_play_set(Ewl_Media *, int)` or stop the video by passing 0 to the same function.

To determine if the video codec allows for seeking in the video you can call `int ewl_media_seekable_get(Ewl_Media *)` which will return 1 if the video is seekable, 0 otherwise. `double ewl_media_position_get(Ewl_Media *)` is used to determine the current position in the video, while `ewl_media_position_set(Ewl_Media *, double position)` can be used to set the position in the video. This value can also be retrieved as a hours, minutes and seconds by calling `ewl_media_position_time_get(Ewl_Media *, int h, int m, double s)`.

If you wish to change the audio settings of the video there are several functions available. These including the ability to get/set the current mute settings: `int ewl_media_audio_mute_get(Ewl_Media *)` and `ewl_media_audio_mute_set(Ewl_Media *, int)`. You can also get/set the volume of the video through the calls: `int ewl_media_audio_volume_get(Ewl_Media *)` and `ewl_media_audio_volume_set(Ewl_Media *, int)`.

Chapter 8. Contributing

If you found this document useful, but lacking in some fashion, please consider contributing back to the document itself. This document is available under an open license and any submissions are greatly appreciated. Any submissions can be sent to zero@perplexity.org [mailto:zero@perplexity.org].

Note that any contributions to this document need to be licensed under the Creative Commons NonCommercial-ShareAlike 1.0 License, which is what this document uses.

If you wish to contribute to the EWL or another part of the EFL, take a look at the www.enlightenment.org [http://www.enlightenment.org] website. All the information on accessing CVS and the mailing lists can be found there.

Thank you.

Appendix A. EWL Media Player Example

Example A.1. Ewl Media Player

```
#include <Ewl.h>

static Ewl_Widget *video;
static Ewl_Widget *fd_win;
static Ewl_Widget *seeker;

typedef struct {
    char *name;
    Ewl_Callback_Function func;
} Control;

void del_cb(Ewl_Widget *w, void *event, void *data) {
    ewl_widget_hide(w);
    ewl_widget_destroy(w);
    ewl_main_quit();
}

void play_cb(Ewl_Widget *w, void *event, void *data ) {
    ewl_media_play_set(EWL_MEDIA(video), 1);
}

void stop_cb(Ewl_Widget *w, void *event, void *data ) {
    ewl_media_play_set(EWL_MEDIA(video), 0);
}

void ff_cb(Ewl_Widget *w, void *event, void *data ) {
    double p = ewl_media_position_get(EWL_MEDIA(video));
    ewl_media_position_set(EWL_MEDIA(video), p + 10.0);
}

void rew_cb(Ewl_Widget *w, void *event, void *data ) {
    double p = ewl_media_position_get(EWL_MEDIA(video));
    ewl_media_position_set(EWL_MEDIA(video), p - 10.0);
}

void video_realize_cb(Ewl_Widget *w, void *event, void *data) {
    double len = ewl_media_length_get(EWL_MEDIA(video));
    ewl_seeker_range_set(EWL_SEEKER(seeker), len);
}

void video_change_cb(Ewl_Widget *w, void *event, void *data) {
    char buf[512];
    int h, m;
    double s;
    Ewl_Text *t = (Ewl_Text *)data;
    double pos = ewl_media_position_get(EWL_MEDIA(video));

    ewl_seeker_value_set(EWL_SEEKER(seeker), pos);
    ewl_media_position_time_get(EWL_MEDIA(video), &h, &m, &s);
    snprintf(buf, sizeof(buf), "%02i:%02i:%02.0f", h, m, s);
    ewl_text_text_set(t, buf);
```

```
}

void seeker_move_cb(Ewl_Widget *w, void *event, void *data) {
    double val = ewl_seeker_value_get(EWL_SEEKER(seeker));
    ewl_media_position_set(EWL_MEDIA(video), val);
}

void fd_win_del_cb(Ewl_Widget *w, void *event, void *data) {
    ewl_widget_hide(w);
    ewl_widget_destroy(w);
}

void open_file_cb(Ewl_Widget *w, void *event, void *data) {
    char *file = NULL;

    ewl_widget_hide(fd_win);
    file = (char *)event;
    if (file)
        ewl_media_media_set(EWL_MEDIA(video), file);
}

void open_cb(Ewl_Widget *w, void *event, void *data ) {
    Ewl_Widget *fd = NULL;

    if (fd_win) {
        ewl_widget_show(fd_win);
        return;
    }

    fd_win = ewl_window_new();
    ewl_window_title_set(EWL_WINDOW(fd_win), "EWL Media Open");
    ewl_window_class_set(EWL_WINDOW(fd_win), "EWL_Media_Open");
    ewl_window_name_set(EWL_WINDOW(fd_win), "EWL_Media_Open");
    ewl_callback_append(fd_win, EWL_CALLBACK_DELETE_WINDOW,
                                            fd_win_del_cb, NULL);
    ewl_widget_show(fd_win);

    fd = ewl_filedialog_new(EWL_FILEDIALOG_TYPE_OPEN);
    ewl_container_child_append(EWL_CONTAINER(fd_win), fd);
    ewl_callback_append(fd, EWL_CALLBACK_VALUE_CHANGED, open_file_cb, NULL);
    ewl_widget_show(fd);
}

void key_up_cb(Ewl_Widget *w, void *event, void *data) {
    Ewl_Event_Key_Up *e = (Ewl_Event_Key_Up *)event;

    if (!strcmp(e->keyname, "p"))
        ewl_media_play_set(EWL_MEDIA(video), 1);

    else if (!strcmp(e->keyname, "s"))
        ewl_media_play_set(EWL_MEDIA(video), 0);

    else if (!strcmp(e->keyname, "q"))
        del_cb(w, event, data);
}

int main(int argc, char ** argv) {
    Ewl_Widget *win = NULL, *o = NULL, *b = NULL;
    Ewl_Widget *controls = NULL, *time = NULL;
    char * file = NULL;

    if (!ewl_init(&argc, argv)) {
        printf("Can't init ewl");
        return 1;
```

```
}

if (argc > 1)
    file = argv[1];

win = ewl_window_new();
ewl_window_title_set(EWL_WINDOW(win), "EWL Media test");
ewl_window_name_set(EWL_WINDOW(win), "EWL_Media_test");
ewl_window_class_set(EWL_WINDOW(win), "EWL_Media_test");
ewl_callback_append(win, EWL_CALLBACK_DELETE_WINDOW, del_cb, NULL);
ewl_callback_append(win, EWL_CALLBACK_KEY_UP, key_up_cb, NULL);
ewl_object_size_request(EWL_OBJECT(win), 320, 280);
ewl_object_fill_policy_set(EWL_OBJECT(win), EWL_FLAG_FILL_ALL);
ewl_widget_show(win);

/* box to contain everything */
b = ewl_vbox_new();
ewl_container_child_append(EWL_CONTAINER(win), b);
ewl_object_fill_policy_set(EWL_OBJECT(b), EWL_FLAG_FILL_ALL);
ewl_widget_show(b);

/* create the time widget now so we can pass it to the video as data */
time = ewl_text_new();
ewl_text_text_set(EWL_TEXT(time), "00:00:00");

/* the video */
video = ewl_media_new(file);
ewl_container_child_append(EWL_CONTAINER(b), video);
ewl_object_fill_policy_set(EWL_OBJECT(video), EWL_FLAG_FILL_ALL);
ewl_callback_append(video, EWL_CALLBACK_REALIZE, video_realize_cb, NULL);
ewl_callback_append(video, EWL_CALLBACK_VALUE_CHANGED, video_change_cb, time);
ewl_widget_show(video);

/* box to contain contols and scrollers */
controls = ewl_vbox_new();
ewl_object_fill_policy_set(EWL_OBJECT(controls),
           EWL_FLAG_FILL_VSHRINK | EWL_FLAG_FILL_HFILL);
ewl_container_child_append(EWL_CONTAINER(b), controls);
ewl_widget_show(controls);

/* hold he controls */
b = ewl_hbox_new();
ewl_container_child_append(EWL_CONTAINER(controls), b);
ewl_widget_show(b);

{
    Control controls [] = {
        { "play", play_cb },
        { "stop", stop_cb },
        { "rewind", rew_cb },
        { "fast forward", ff_cb },
        { "open", open_cb },
        { NULL, NULL }
    };
    int i;

    for(i = 0; controls[i].name != NULL; i++) {
        o = ewl_button_stock_new(controls[i].name);
        ewl_container_child_append(EWL_CONTAINER(b), o);
        ewl_callback_append(o, EWL_CALLBACK_CLICKED,
                                  controls[i].func, NULL);
        ewl_widget_show(o);
    }
}
```

```
    b = ewl_hbox_new();
    ewl_container_child_append(EWL_CONTAINER(controls), b);
    ewl_widget_show(b);

    /* the video seeker */
    seeker = ewl_seeker_new(EWL_ORIENTATION_HORIZONTAL);
    ewl_container_child_append(EWL_CONTAINER(b), seeker);
    ewl_object_fill_policy_set(EWL_OBJECT(seeker),
                        EWL_FLAG_FILL_VSHRINK | EWL_FLAG_FILL_HFILL);
    ewl_seeker_value_set(EWL_SEEKER(seeker), 0.0);
    ewl_seeker_range_set(EWL_SEEKER(seeker), 0.0);
    ewl_seeker_step_set(EWL_SEEKER(seeker), 1.0);
    ewl_callback_append(seeker, EWL_CALLBACK_VALUE_CHANGED, seeker_move_cb, NULL);
    ewl_widget_show(seeker);

    /* the time text spot */
    ewl_container_child_append(EWL_CONTAINER(b), time);
    ewl_object_insets_set(EWL_OBJECT(time), 0, 3, 0, 0);
    ewl_object_fill_policy_set(EWL_OBJECT(time), EWL_FLAG_FILL_SHRINK);
    ewl_widget_show(time);

    ewl_main();
    return 0;
}
```