EDJE PROGRAMMING GUIDE

Aug. 2025

EDC File Basics

Let's start with a “Hello World” example. This is going to be in verbose sytax to make it as easy as possible to break down before we start making it more compact. Start with a source “EDC” file like below:

example.edc (/_export/code/docs/themes/start?codeblock=0)

```
collections {
  group {
    name: "example";
    parts {
      part {
        name: "background";
        type: RECT;
        description {
          state: "default";
          color: 64 64 64 255;
        }
      }
      part {
        name: "label";
        type: TEXT;
        description {
          state: "default";
          color: 255 255 255 255;
          text {
            font: "Sans";
            size: 10;
            text: "Hello World";
          }
        }
      }
    }
  }
}
```

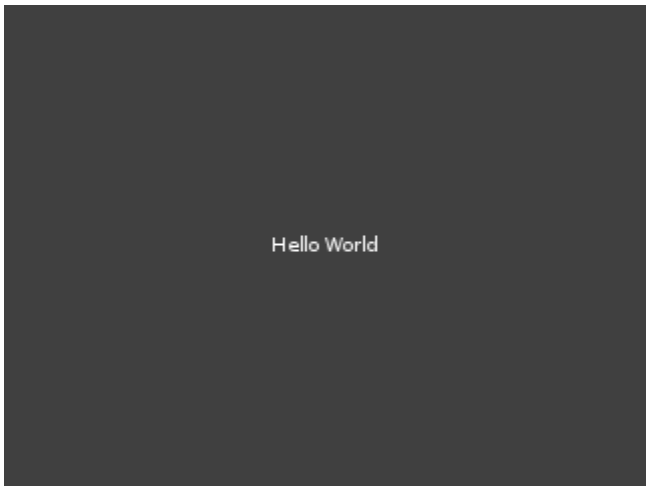
Compiling it is easy. The compiled EDJ file is portable. It can be used on any OS (Operating System) and any architecture. It's basically a structured archive of all of the source material packed into a single file that is compressed and usable “as-is” at runtime without installation or unpacking.

```
edje_cc example.edc
```

Now a quick way to see your result is with the `edje_player` tool that will load a specific group in a file or just use the first it finds. In our case here we have only a single group so it should load just this group within the collections and let us see it:

```
edje_player example.edj
```

You now should see something like this in a window:



The window will be resizable, so resize it to see what happens when your group is resized too. One of the key ideas behind Edje is to be able to make both resizable AND scaleable components that can adapt to sizing needs and UI/DPI scaling needs.

The Anatomy of an Edje File

An Edje file is compiled from a combination of source “EDC” files and other data (images, fonts, sound samples, ...) into a single stand-alone binary “EDJ” file that is used at runtime. These files are, by default, de-compileable again back to source with the `edje_decc` tool. These files do not execute code, and are system-independent. That may use various compression schemes based on options to `edje_cc`. The source EDC files are passed through a C pre-processor so they can use CPP directives such as `#include`, `#define`, `#ifdef`, the macros defined by `#define`, etc. and this can be used to make your EDC files more maintainable by being able to generate content from smaller macros or included files, split your content up into many files that you `#include` together and so on. But let's get into the actual structure.

An EDC file will look a bit like a mix of JSON and a C language structure definition. Everything has some kind of parent/child hierarchy with parent sections containing child sections and so on. It is also possible to skip a section and use periods (.) to declare a parent and child relationship directly on a line, so You could express:

example.edc (/_export/code/docs/themes/start?codeblock=3)

```
text {
  font: "Sans";
  size: 10;
  text: "Hello World";
}
```

As

example.edc (/_export/code/docs/themes/start?codeblock=4)

```
text.font: "Sans";
text.size: 10;
text.text: "Hello World";
```

You can also remove whitespace before keywords and after statement delimiters (`;`), so the above could also be done expressed as:

example.edc (/_export/code/docs/themes/start?codeblock=5)

```
text { font: "Sans"; size: 10; }
text.text: "Hello World";
```

You should take advantage of white-space compression to make your EDC files less lengthy and thus easier to read and maintain. So let's take the above hello world example and make it a bit more compact before we continue:

example.edc (/_export/code/docs/themes/start?codeblock=6)

```
collections {
  group { name: "example";
    parts {
      part { name: "background"; type: RECT;
        description { state: "default";
          color: 64 64 64 255;
        }
      }
      part { name: "label"; type: TEXT;
        description { state: "default";
          color: 255 255 255 255;
          text { font: "Sans"; size: 10; }
          text.text: "Hello World";
        }
      }
    }
  }
}
```

We will use this kind of white-space compression from now on to keep things more compact. Remember that the same declaration is able to be expressed in many different ways, so don't get confused when you see the same thing expressed with different white space or with different parent/child syntax as they are equivalent.

Edje files generally contain several major sections. `collections` is the section where a series of `group` sections are declared. Each group is accessible by its `name` via Edje or Elementary API (Application Programming Interface). A single EDJ file may contain 0, 10, 100, or 1000 groups ... or more. It depends on how much you pack in. The names of such groups is a free-form string, but generally there is a convention to use something like a file path so `group/children/thing` would be common. So a very basic EDC source file may begin as:

example.edc (/_export/code/docs/themes/start?codeblock=7)

```
collections {
  group { name: "example";
  }
}
```

We have a group in our collections, and this group has nothing in it. What exactly is a `group` then? It's a group of child objects, programs that respond to signals/events and other layout logic that define a single graphical element. Code can ask the Edje library to use this defined group from that file such as:

example.c (/_export/code/docs/themes/start?codeblock=8)

```
obj = elm_layout_add(win);
elm_layout_file_set(obj, "example.edj", "example");
elm_win_resize_object_add(win, obj);
evas_object_show(obj);
```

Elementary and Enlightenment have code like this everywhere (or the lower level versions of it). Almost every Elementary widget actually creates such layout/Edje objects internally and sources them from theme files such as `default.edj` that ships with EFL. Every single widget will likely go back to this file and figure out which bit of data maps to this name, load it and instantiate the objects based on the design data in the EDJ file. This is mostly transparent to a user or programmer, other than when they wish to change the look of their UI ... they then can make their own themes and override the standard search for a theme with their own.

This is how any application may provide a custom look/feel of its own (it's own Themes), as well as added layout elements that are not normal widgets, built out of such data files and much much more. Enlightenment even will only accept EDJ files as wallpapers because then it then doesn't need to know how to tile, scale or otherwise lay out the wallpaper content. Edje takes care of this. The Wallpaper import dialog is simply generating a sample EDC file that includes the selected image file with the layout options selected. An added bonus is that a single EDJ file can include multiple resolutions of an image to optimally load the best one, scale and adapt on the fly and even animate.

So learning how Edje works is the key to the kingdom of Enlightenment. Master Edje and your UI can be changed from clean modern flat styles through to skeumorphic madness and anything else your imagination can come up with. So let's continue with the basic anatomy.

Every `group` will likely have `parts`, possibly `programs` and may even have it's own `images` section as well as a few others. The `parts` section defines a series of parts in stacking order from bottom to top that comprise the group object. So a small step forward with our example may become something like this:

example.edc (/_export/code/docs/themes/start?codeblock=9)

```
collections {
  group { name: "example";
    parts {
      part { name: "background"; type: RECT;
      }
      part { name: "label"; type: TEXT;
      }
    }
  }
}
```

We have 2 parts, one named “background” that is of type RECT (a basic rectangle) and one “label” that is of type TEXT (a simple single line text object with no markup etc.). You generally will want names for any part you wish to later access or address by changing its state or otherwise accessing it from outside Edje (i.e. from the calling application). Come up with names that are memorable. In EFL we generally namespace our names. If the part name is meant to be accessed from outside the group (e.g. from an app), then we will namespace something like “e.text” or “elm.text” etc. but using a dot (`.`) with a prefix of the namespace then following namespaces. A sign that a part is “official” and is expected to exist or be interacted with is that it has a dot in its name with a namespace.

But these parts are pretty useless as Edje has no idea of their description - what color are they? What Font used? What sized font? We need to provide at LEAST a `default` description.

example.edc (/_export/code/docs/themes/start?codeblock=10)

```
collections {
  group { name: "example";
    parts {
      part { name: "background"; type: RECT;
        description { state: "default";
          color: 64 64 64 255;
        }
      }
      part { name: "label"; type: TEXT;
        description { state: "default";
          color: 255 255 255 255;
          text { font: "Sans"; size: 10; }
          text.text: "Hello World";
        }
      }
    }
  }
}
```

And now we have a basic example. The rectangle is a dim grey (color is in R G B A with values from 0 to 255 for the elements) normally, but you can also use html-style color notations like “#404040ff” instead of 64 64 64 255 like `color: "#404040ff";` . We have a single line text label that uses the “Sans” font at size 10, is white in color and displays the text “Hello World” by default.

For a pretty complete Edje programming guide, please see:

Edje Programming Guide (/program_guide/edje_pg)

Programming Guide Menu

Edje Programming Guide

This programming guide shows you how to write an EDC file that can be used to theme a EFL application. It describes concepts about parts positioning and resizing. It also explains part animations that can be done through programs.

Table of Contents

Related Info

Quick How-tos

Adding Element on Screen

Add a new part inside the parts block (/develop/legacy/program_guide/edje/part_block).

Using Image

List the image in the images block (/develop/legacy/program_guide/edje/edje_blocks#Images_Block), make sure the part (/develop/legacy/program_guide/edje/part_block) has type “IMAGE” and set the normal property inside description.image (/develop/legacy/program_guide/edje/part_block#Image) of part.

Using Same Color Definitions Across Multiple Elements

Define a color class (/develop/legacy/program_guide/edje/edje_bloks#Color_classes_Block) and set the description.color_class (/develop/legacy/program_guide/edje/part_block#Description) property.

Positioning or Resizing a Part (Relative and Absolute Positioning)

Fill in the rel1 and rel2 (/develop/legacy/program_guide/edje/part_block#Rel1rel2) structures inside the description block (/develop/legacy/program_guide/edje/part_block#Description).

Hiding Part

Set the visible property (/develop/legacy/program_guide/edje/part_block#Description) to 0.

Animating Part

Create several description blocks (/develop/legacy/program_guide/edje/part_block#Description) inside your part (/develop/legacy/program_guide/edje/part_block) and give each of them a different value for state. Set one description for the initial state and one for the end state.

Create a program (/develop/legacy/program_guide/edje/program#Program) with an action that is STATE_SET “end_state” 0.0; and with a target that is the name of the part. You can also set a non-default transition.

When defining the second description (/develop/legacy/program_guide/edje/part_block#Description), inherit from the first part in order to re-use the values which are already defined.

The after property of the program block (/develop/legacy/program_guide/edje/program#Program) is used to trigger another program after the animation is done. It can be used to trigger another animation or to emit a signal to the C part of the program.

Making Genlist Item Theme

Create a Group Block (/develop/legacy/program_guide/edje/group_block) with one part element (/develop/legacy/program_guide/edje/part_block) for each part that can be filed from the C code and set the items properties inside the group element:

```
items: "texts" "text_part_1 text_part_2";
items: "icons" "image_part_1 image_part_2";
```

On the C side, the `text_get` and `content_get` callback are called respectively with `text_part_1` and `text_part_2`, and `image_part_1` and `image_part_2`.

Basic Concept

Table of Contents

What is an EDC File?

An EDC file stands for Edje data collection. It is a text file that contains special code format describing the position, size, and other parameters of graphical elements that compose the visual aspect of your application. In addition to graphical elements, it can also handle sounds.

The syntax for the Edje data collection files follows a simple structure of “blocks { .. }” that can contain “properties: ..”, more blocks, or both.

An EDC file has the “.edc” extension.

Compiling EDC File

EDC file needs to be compiled into a “.edj” file using Edje library tools. After compiling the “.edj” file can be used by a EFL/elmentary application.

Here is an example about compiling `helloworld.edc` to “.edj” file using `edje_cc tool` :

```
$ edje_cc helloworld.edc
```

This command creates a `helloworld.edj` file.

An EDC file can use external files such as sounds, images, or fonts. The path to these resources are passed to the `edje_cc` tool so that they are included in the final “.edj” file.

```
$ edje_cc -sd $SOUNDS_DIR -fd $FONTS_DIR -id $IMAGES_DIR
```

`SOUNDS_DIR` , `FONTS_DIR` , and `IMAGES_DIR` are the paths for sounds, fonts, and images resources respe[illegible].

Writing Simple EDC File

The code example below shows you the structure of an EDC file. It is a collection of groups that contain parts and programs.

```
collections
{
   group
   {
      name: "my_group";
      parts {}
      programs {}
   }
}
```

Groups are identified with a name, parts correspond to the graphical elements. Each one of them can have several states that describe a specific position, size, and visual aspect. Programs contain the program code, such as interaction with the main application through signals. Also animations are defined here (changing a part state using an animated transition).

The description field is where the state of a part is written.

```
part
{
   description
   {
      state: "default" 0.0;
   }
   description
   {
      state: "state1" 0.0;
   }
   description
   {
      state: "state2" 0.0;
   }
}
```

As an example, here is a simple EDC file that contains only one part and one program. The part is a rectangle with blue state and red state, the program changes the state from blue to red when user clicks on the rectangle.

```
collections
{
   group
   {
      name: "example";
      parts
      {
         // create the part
         part
         {
            name: "rectangle";
            // set the type to RECT (rectangle)
            type: RECT;
            // default state (blue color)
            description
            {
               state: "default" 0.0;
               align: 0.0 0.0;
               // blue color
               color: 0 0 255 255;
            }
            // second state (red color)
            description
            {
               state: "red" 0.0;
               align: 0.0 0.0;
               // red color
               color: 255 0 0 255;
            }
         }
      }
      programs
      {
         // create a program
         program
         {
            name: "change_color";
            // program is triggered on mouse click
            signal: "mouse,clicked,*";
            source: "*";
            // set the red state of the "rectangle" part
            action: STATE_SET "red" 0.0;
            target: "rectangle";
         }
      }
   }
}
```

A program is triggered when receiving a signal from a specific source (here all the sources are taken into account). When launched, it does the action (changing the state of a part) on the target (the rectangle).

Animating Theme Using Programs

The previous example showed how to change the state of a part. It is also possible to use the transition parameter to create an animation between the 2 states. You can specify a transition type (ACCELERATE, DECELERATE, SINUSOIDAL, LINEAR, ...) and length (in seconds) of the transition.

The following code example animates the previous state change using a linear transition of 2 seconds.

```
programs
{
   program
   {
      name: "change_color";
      signal: "mouse,clicked,*";
      source: "*";
      action: STATE_SET "red" 0.0;
      target: "rectangle";
      transition: LINEAR 2.0;
   }
}
```

Edje calculates all frames needed for the animation. The result is a smooth animation between the two states and it takes 2 seconds.

Positioning Basic Parts

Size of a part (in pixels) is set using the min and max parameters. The following code example sets the minimum and maximum size of the rectangle part to 200×200 px.

```
part
{
   name: "rectangle";
   type: RECT;
   description
   {
      state: "blue" 0.0;
      align: 0.0 0.0;
      // set the size to 200x200
      min: 200 200;
      max: 200 200;
      // blue color
      color: 0 0 255 255;
   }
}
```

Position of the parts is defined in the `rel1` and `rel2` blocks. `rel1` and `rel2` blocks are used to define

respectively the upper-left corner and the lower-right corner of the part. Position can be defined relatively to other parts (with the relative parameter) as an offset (offset parameter). When using relative positioning, the `to`, `to_x` and `to_y` parameters are used to define to which part the relative positioning is done. If nothing else is specified, the positioning is relative to the parent's part.

To demonstrate the relative positioning, here is a code example that creates another part and positions it under the first part (the upper-left corner of the new part will start at the lower-left corner of the previous one).

```
part
{
   name: "rectangle2";
   type: RECT;
   description
   {
      state: "green" 0.0;
      align: 0.0 0.0;
      // set the size to 200x200
      min: 200 200;
      max: 200 200;
      // green color
      color: 0 255 0 255;
      // set the position
      // rel1 is relative to "rectangle"
      rel1
      {
         relative: 0.0 1.0;
         to: "rectangle";
      }
      // rel2 is relative to the parent
      rel2
      {
         relative: 1.0 1.0;
      }
   }
}
```

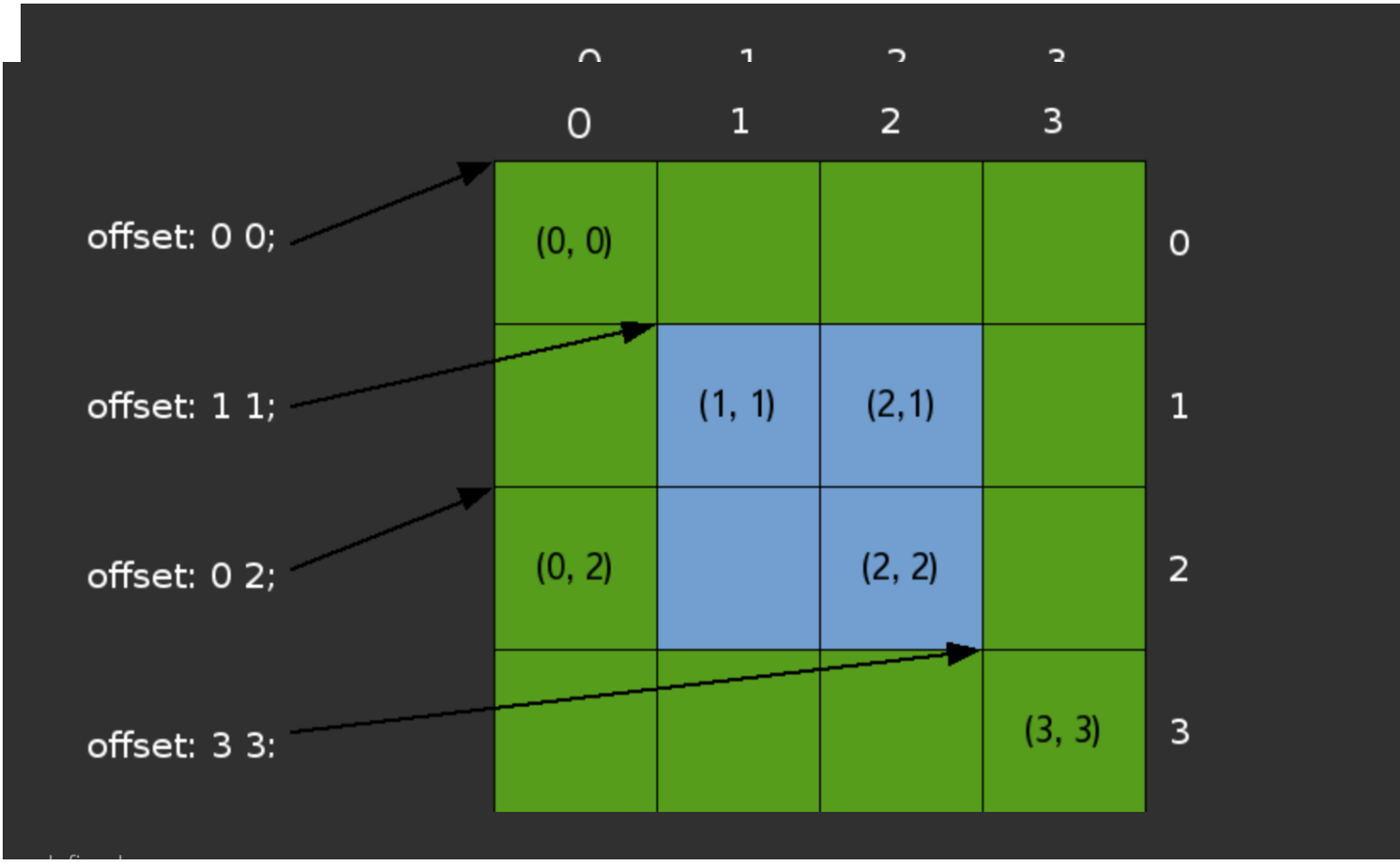
> The `align` parameter defines how the parts align themselves in the main window if their size is smaller than the main window. If nothing is specified, the parts are aligned in the center of the window.

Adding Offset to Relative Positioning

The `rel1` and `rel2` structures also support offset which is a complement to the relative positioning: the corners are first placed according to their relative parameters and then adjusted using the offsets.

The picture below shows the pixel positions for a 4×4 rectangle. The indices start in the top-left corner at (
increase to the right and to the bottom. Since the indices have started at 0, the 4th pixel has an index of 3.

Therefore, in order to create a 2×2 blue rectangle centered inside that green square, the top-left corner has to be (1, 1) and the bottom-right one has to be (2, 2).



Edje needs the following things defined:

- the part coordinates depending on the size and position of the green rectangle
- the relative component of positions is the same: the top-left corner of the green rectangle
- the top-left pixel is (1, 1) and the bottom-right one is (2, 2)

The following code example defines these things:

```
name: "blue rectangle";

rel1.to: "green rectangle";
rel1.relative: 0 0;
rel1.offset: 1 1;

rel2.to: "green rectangle";
rel2.relative: 0 0;
rel2.offset: 2 2;
```

For most tasks, relative positioning is simpler than using offsets. Offsets are usually left for fine-tuning and creating borders.

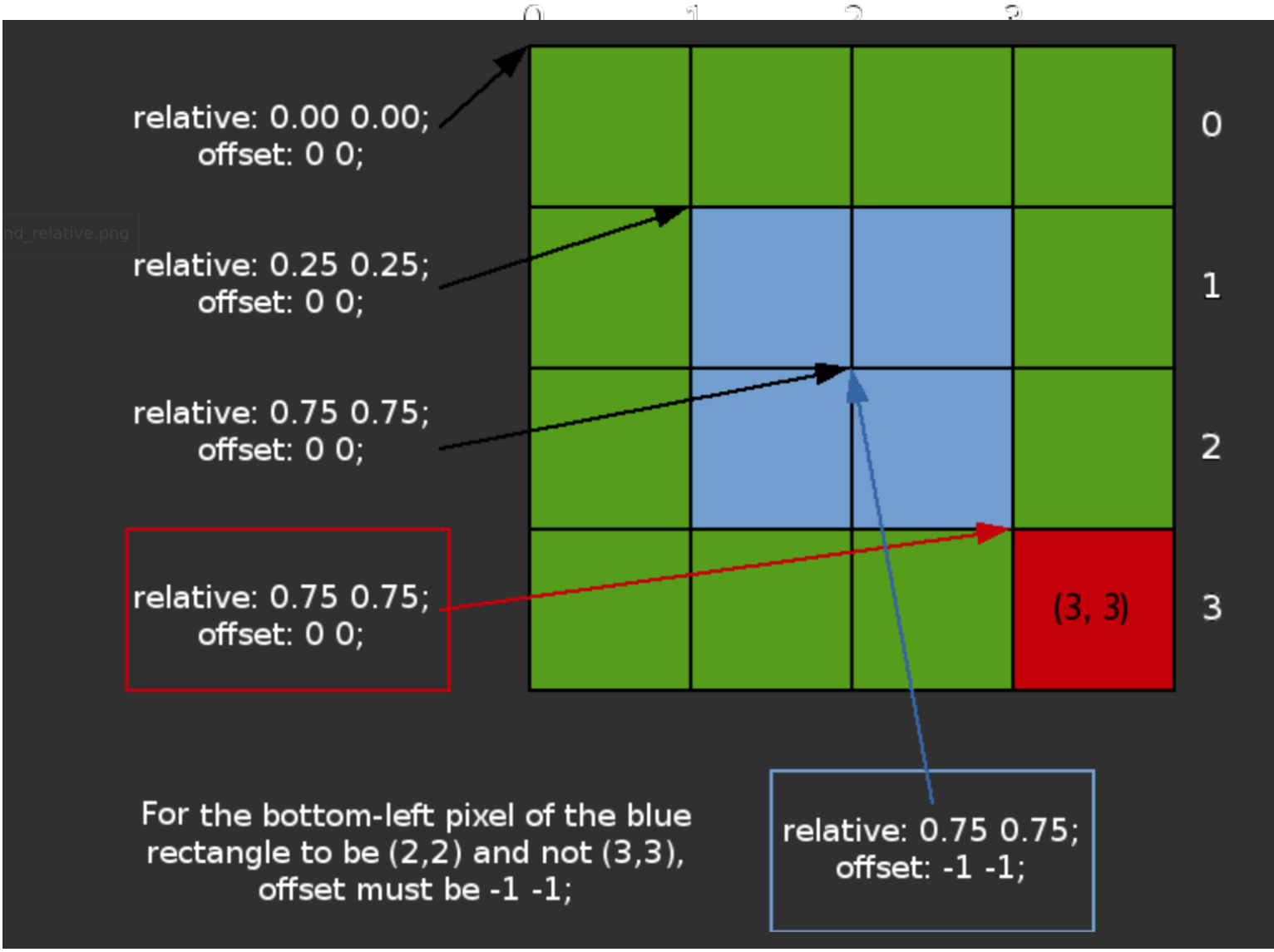
The example below is similar to the previous one but uses relative positioning instead of offsets to achiev equivalent at 4×4 but could scale to larger sizes.

The blue square starts at 25% of the green square (both vertically and horizontally) and ends at 75% of it (again, both vertically and horizontally).

Just like in the previous example, the blue rectangle is named and Edje is told what the object of reference is:

```
name: "blue rectangle";
rel1.to: "green rectangle";
rel2.to: "green rectangle";
```

The image below shows how to refer pixels using relative positioning when the offsets are (0, 0).



(/_detail/edje_rel1_rel2_offsets_and_relative.png?id=develop%3Alegacy%3Aprogram_guide%3Aedje%3Abasic_concepts)

Note the addressing of pixels: (0, 0) is addressed through `relative: 0 0; offset 0 0;` and each additional 0.25 in the relative field gives a 1-pixel move. With this, the pixel addressed through `relative: 0.75 0.75; offset: 0 0;` is at (3, 3) and not (2, 2)!.

This comes from a design choice in Evas and Edje which favor simplicity. In the examples shown in this gu

there are 4 pixels and therefore when the `[0; 1)` range is divided in 4, the result is `[0; 0.25), [0.25; 0.50), [0.50; 0.75), [0.75; 1.00)`. With Edje, the value used to refer to each segment is the left bound and therefore, 0.75 refers to `[0.75; 1.00)`, i.e. the bottom-right pixel of the green rectangle and not the 3/4th one.

The way to refer to the pixel right before is to set the `rel2` bound to `relative: 0.75 0.75;`, as would be expressed naturally, and `offset: -1 -1;`. This can also be understood as extending the rectangle up to 75% of its parent with the upper bound excluded (as shown in the `[0.50; 0.75)`).

Since `-1 -1` is the most common offset wanted for `rel2`, it is the default value; i.e. the default behavior is practical.

Calculating Edje Object Total Size

When the EDC file is composed of a lot of parts, Edje calculates the size of the global Edje object, by taking all the parts and their parameters into account. Some parameters have an role in this calculation and affect the global size:

- `min` and `max` : these define the minimum and the maximum size of a part.
- `rel1` and `rel2` : these specify the relative position of a part.
- `align` : this relates to the alignment of the part in the parent's object.
- `fixed` : this defines if the part has a fixed size.

> `fixed` parameter can only be used on `TEXTBLOCK` type parts. Setting this parameter to `fixed: 1 1` will not take into account the part for the calculation of the global size.

Using Edje Size Hints

Any `Evas_Object` can have hints, so that the object knows how to properly position and resize. Edje uses these hints when swallowing an `Evas_Object` to position and resize it in the `SWALLOW` part of the EDC file.

Size hints are not a size enforcement, they just tell the parent object the desired size for this object. Then, the parent tries to get as close as possible to the hint.

Hints are set in an `Evas_Object` using the `evas_object_size_hint_*()` functions.

Min Size Hint

This sets the hints for the object's minimum size, given in pixels.

Here the horizontal and vertical min size hints of an `Evas_Object` are set to 0 pixels.

```
Evas_Object *object;
evas_object_size_hint_min_set(object, 0, 0);
```

Max Size Hint

This sets the hints for the object's maximum size, given in pixels.

Here the horizontal and vertical max size hints of an `Evas_Object` are set to 200 pixels.

```
evas_object_size_hint_max_set(object, 200, 200);
```

Request Size Hint

This sets the hints for the object's optimum size.

The following code example defines that the optimum size of a part is 200×200 pixels.

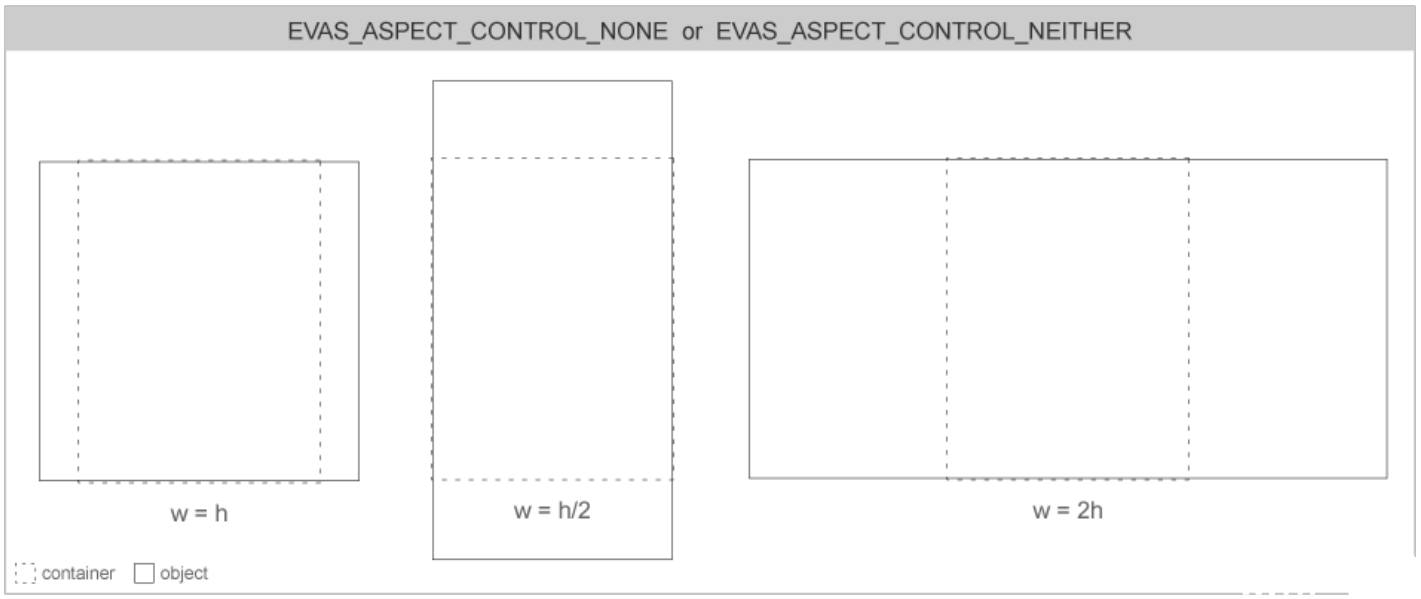
```
evas_object_size_hint_request_set(object, 200, 200);
```

Aspect Size Hint

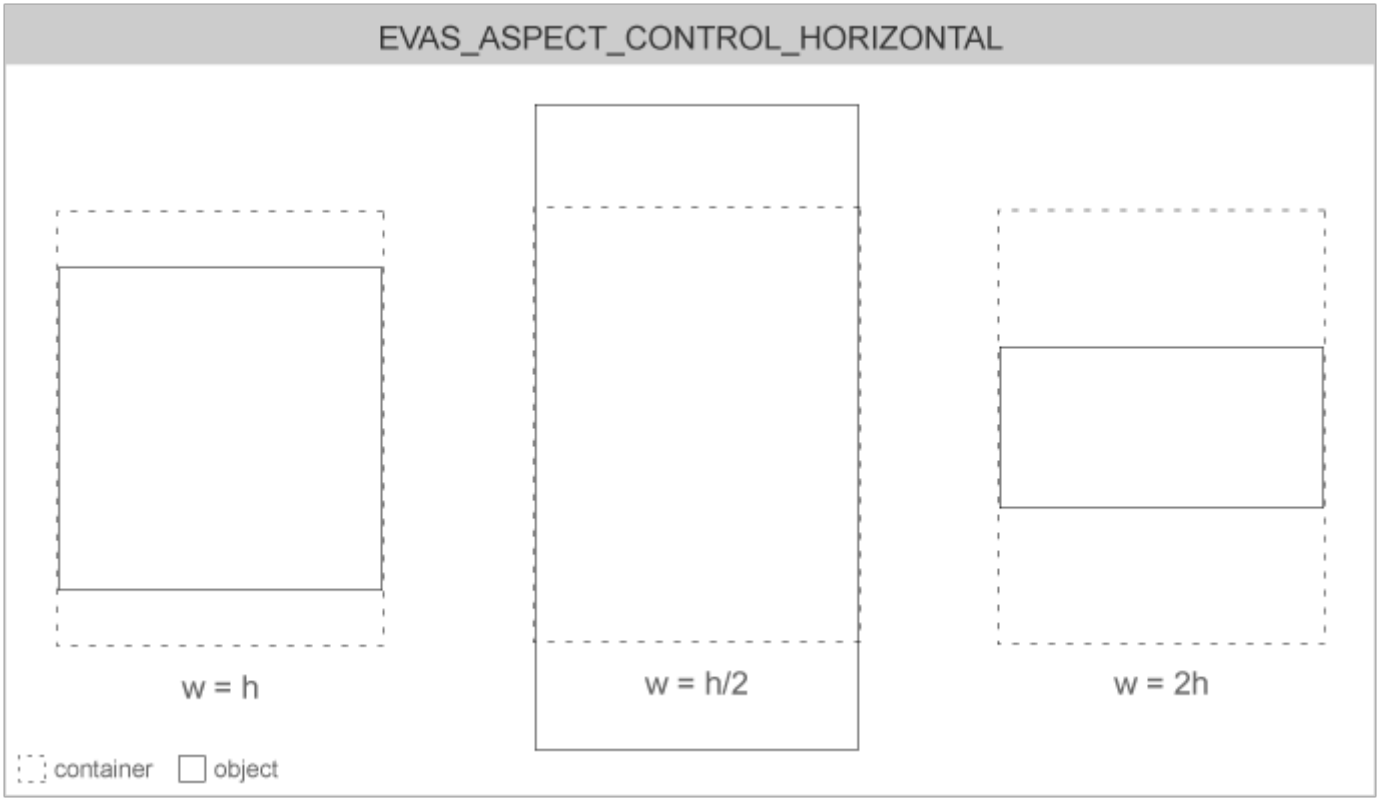
This sets the hints for the object's aspect ratio. Available aspect size hints are:

- `EVAS_ASPECT_CONTROL_NONE`
- `EVAS_ASPECT_CONTROL_HORIZONTAL`
- `EVAS_ASPECT_CONTROL_VERTICAL`
- `EVAS_ASPECT_CONTROL_BOTH`

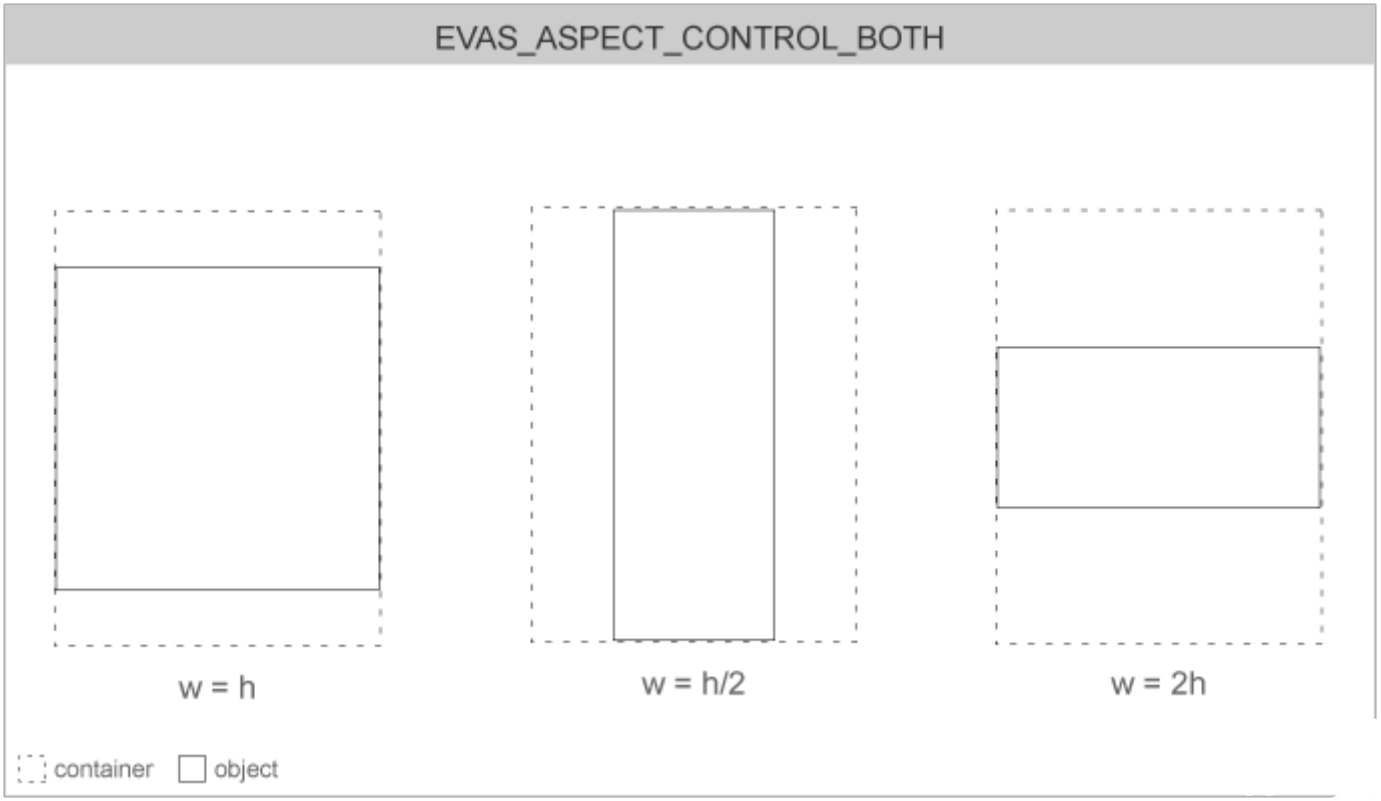
The other parameters are aspect width and height ratio. These integers are used to calculate the proportions of the object. If aspect ratio terms are null, the object's container ignores the aspect and scale of the object and occupies the whole available area.



(/_detail/edje_aspect-control-none.png?id=develop%3Alegacy%3Aprogram_guide%3Aedje%3Abasic_concepts)



(/_detail/edje_aspect-control-horizontal.png?
id=develop%3Alegacy%3Aprogram_guide%3Aedje%3Abasic_concepts)



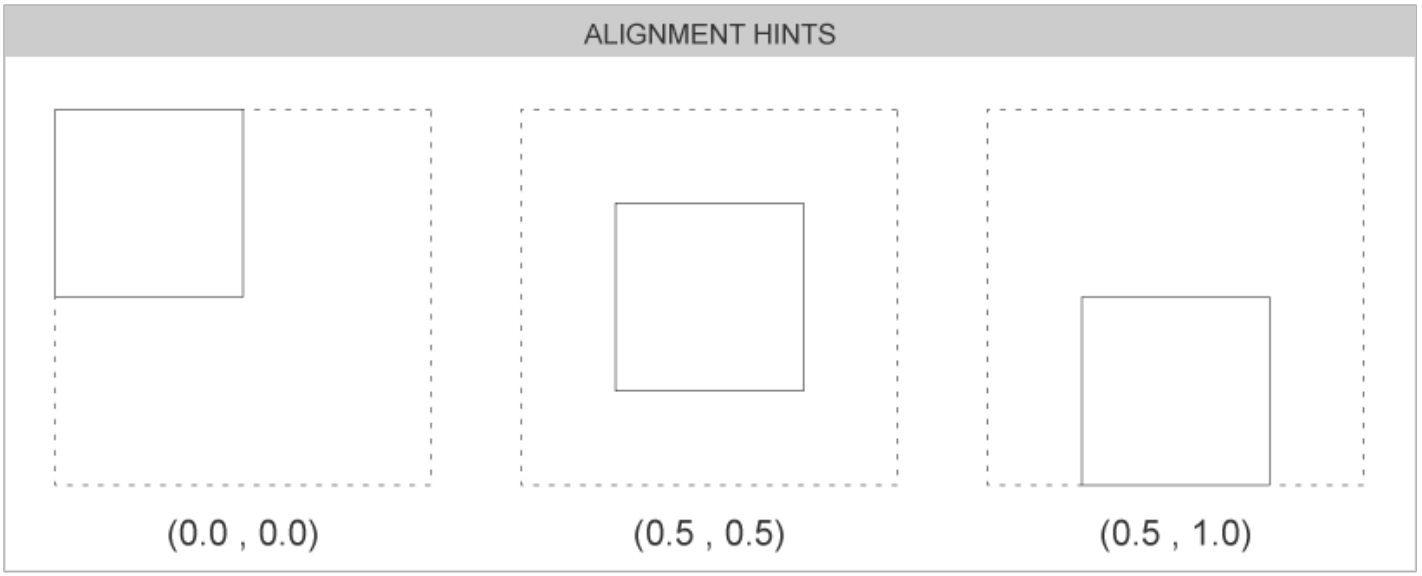
(/_detail/edje_aspect-control-both.png?id=develop%3Alegacy%3Aprogram_guide%3Aedje%3Abasic_concepts)

The following code example sets the aspect size hint to `EVAS_ASPECT_CONTROL_BOTH` with a width of 100 and a height of 200. So aspect ratio should be 1/2.

```
evas_object_size_hint_aspect_set(object, EVAS_ASPECT_CONTROL_BOTH, 100, 200);
```

Align Size Hint

This sets the hints for the object's alignment. This hint is used when the object size is smaller than its parent's. The special `EVAS_HINT_FILL` parameter uses maximum size hints with higher priority, if they are set. Also, any padding hints set on objects are added up to the alignment space on the final scene composition.



(/_detail/edje_align_hints.png?id=develop%3Alegacy%3Aprogram_guide%3Aedje%3Abasic_concepts)

In the code below, the special `EVAS_HINT_FILL` parameter is used.

```
evas_object_size_hint_align_set(object, EVAS_HINT_FILL, EVAS_HINT_FILL);
```

Weight Size Hint

This sets the hints for the object's weight. The weight tells to a container object how the given child is resized. Using `EVAS_HINT_EXPAND` parameter asks to expand the child object's dimensions to fit the container's own.

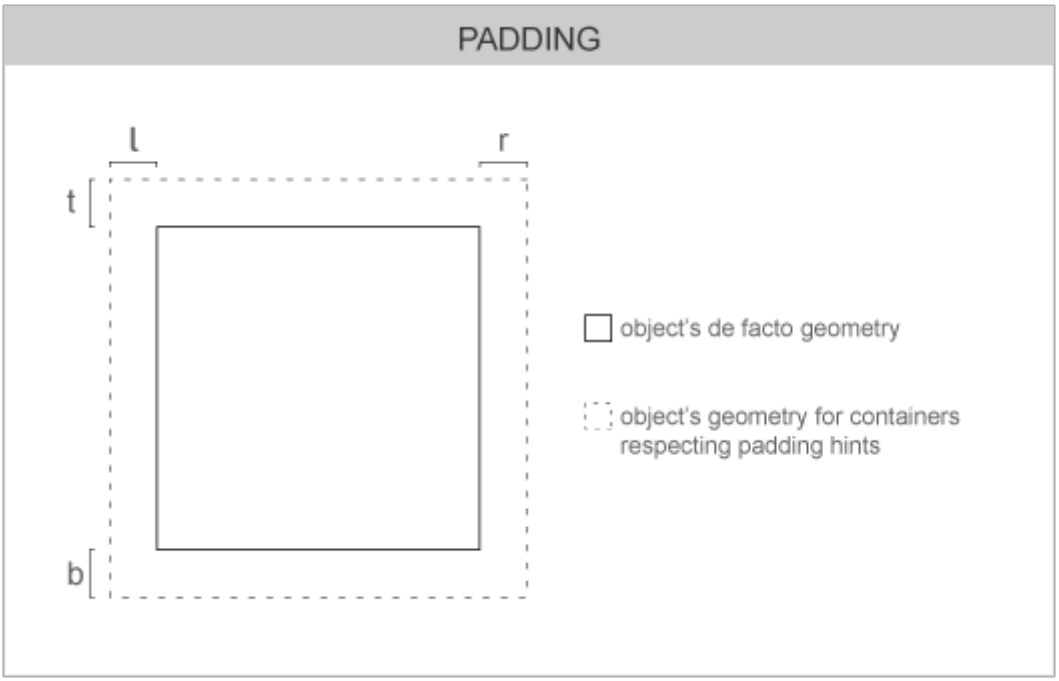
When several child objects have different weights in a container object, the container distributes the space it has to layout them by those factors. Most weighted children get larger in this process than the least ones.

Here the container is asked to expand the object in both directions.

```
evas_object_size_hint_weight_set(object, EVAS_HINT_EXPAND, EVAS_HINT_EXPAND);
```

Padding Size Hint

This sets the hints for the object's padding space. Padding is extra space an object takes on each of its delimiting rectangle sides. The padding space is rendered transparent. Four hints can be defined, for the left, right, top, and bottom padding.



(/_detail/edje_padding-hints.png?id=develop%3Alegacy%3Aprogram_guide%3Aedje%3Abasic_concepts)

Here the padding hints are set to 5 pixels on each side of the object.

```
evas_object_size_hint_padding_set(object, 5, 5, 5, 5);
```

Scaling Objects

Table of Contents

Part Scaling

When scaling an Edje object, only the parts that are declared scalable in the EDC file follow the scale request. This is done using the “scale” parameter.

As an example, the previous “rectangle2” part is set as scalable, so that it is scaled when the Edje object is scaled.

```
part
{
   name: "rectangle2";
   type: RECT;
   scale: 1;
   description
   {
      state: "green" 0.0;
   }
}
```

Using Image Set

When using images in the Edje EDC file, image file names must be listed in a images block, so that Edje can use them in the theme. In this block, the compression level and compression method of the image can also be defined.

Here is an example of an images block that lists two image files.

```
images
{
   image: "image1.jpg" COMP;
   image: "image2.png" LOSSY 99;
}
```

As the devicescan have different screen sizes and resolutions, it is possible to define that the used image set

depends on the resolution.

In the example below the image named “image3” selects different files depending on image size.

```
images
{
   set
   {
      name: "image3";
      image
      {
         image: "image3_1.png" LOSSY 90;
         size: 0 0 50 50;
      }
      image
      {
         image: "image3_2.png" LOSSY 90;
         size: 51 51 200 200;
      }
      image
      {
         image: "image3_3.png" LOSSY 90;
         size: 201 201 500 500;
      }
   }
}
```

This is done with the “size” attribute that specifies the minimal width and height and maximal width and height. If the width and height of the object are below 50px, `image3_1.png` picture file is used. If they are bigger than 201px, `image3_3.png` file is used.

Resizing Borders

Sometimes it is needed to keep the borders of an image intact when resizing or scaling it. The `border` attribute is here to do that.

```
border [left] [right] [top] [bottom]
```

If set, the area (in pixels) of each side of the image is displayed as a fixed size border, from the side -> inwards. This prevents the corners from being changed on a resize.

Here is a code example how to set a border of 10px on each side of the image. This 10px border will not be resized when scaling the image.

```
part
{
   name: "image_border";
   type: IMAGE;
   scale: 1;
   description
   {
      image
      {
         normal: "image1.jpg";
         border: 10 10 10 10;
      }
   }
}
```

Edje Swallow

The parent of all layout widgets is Edje. Edje as explained in the Edje Parts and Blocks (/develop/legacy/program_guide/edje/edje_parts_and_blocks) using text description. One of its main features is the possibility to create “Swallows” objects. When this kind of object is added in an Evas, it contains any other `Evas_Object`. To create a swallow part, create first the EDC file:

```
collections
{
   group
   {
      name: "container";
      parts
      {
         part
         {
            name: "part.swallow"
            type: SWALLOW;
            description
            {
               state: "default" 0.0;
               rel1.offset: 31 31;
               rel2.offset: -32 -32;
            }
         }
      }
   }
}
```

This EDC describes a group named “container”, with one part inside of type SWALLOW and with the name `part.swallow`. This part is centered inside the parent (it is the default behavior) but there are 32 pixels free all around this part. Use `edje_cc` to compile the EDC file into a binary EDJ file:

```
edje_cc -o container.edj container.edc
```

Create an Edje object and load this file:

```
edje = edje_object_add(evas_object_evas_get(parent));
edje_object_file_set(edje, "container.edj", "container");
```

`edje_object_add` as opposed as all elementary object, does not take an `Evas_Object` as a parent. Give it the Evas on which the object is added. As the

parent is already added on an Evas by elementary, retrieve a reference on it by using the `evas_object_evas_get` API (Application Programming Interface).

`edje_object_file_set` is used to set the Edje file from which the object is loaded. The object itself is the name of the group as defined in the EDC file, in this case it is “container”.

Use the API (Application Programming Interface) `edje_object_swallow` to swallow any kind of `Evas_Object` inside.

```
ic = elm_icon_add(parent);
elm_image_file_set(ic, "c1.png", NULL);
edje_object_part_swallow(edje, "part.swallow", ic);
```

The `elm_image_file_set()` function parameters are linked to Edje. The second argument in this example is a PNG file; however, it can also be an Edje file. In that case, the third argument must be the Edje group to load, exactly as previously shown with the `edje_object_file_set()` function.

Create complex layout for your application with Edje. It is may not be the most easy way, but it is the most powerful. This Edje layout is used all around elementary and is the basis of the layout widget.

ELM Layout

Layout is a container widget. The basic use of `elm_layout`, with default style is already documented in the Container guide (/develop/legacy/program_guide/containers_pg). Elm layout takes a standard Edje design file and wraps it in a widget. Layouts are the basis of graphical widgets which are used in Elementary.

Table of Content

Adding Layout

Create a new elementary layout using `elm_layout_add`:

```
Evas_Object *layout;
layout = elm_layout_add(parent);
```

As for Edje swallows, load an Edje file. Create first an Edje file, that contains a black rectangle and an icon in the center.

```
images
{
   image: "c1.png" COMP;
}

collections
{
   group
   {
      name: "my_layout";
      parts
      {
         part
         {
            name: "background";
            type: RECT; description
            {
               state: "default" 0.0; color: 0 0 0 255;
            }
         }
         part
         {
            name: "background";
            type: IMAGE;
            description
            {
               state: "default" 0.0;
               rel1.offset: 31 31;
               rel2.offset: -32 -32;
               default.image: "c1.png";
            }
         }
      }
   }
}
```

Compile it with `edje_cc -o edje_example.edj edje_example.edc`.

This file can be loaded with `elm_layout_file_set`:

```
elm_layout_file_set(layout, "edje_example.edj", "my_layout");
```

The layout widget may contain as many parts/children as described in its theme file (EDC). Some of these children can have special types:

- `SWALLOW` (content holder)
- `BOX`
- `TABLE`

Only one object can be added to a `SWALLOW`. The `elm_layout_content_set()/get/unset` functions are used to manage objects in a `SWALLOW` part. After being set to this part, the object's size, position, visibility, clipping

and other description properties are controlled by the description of the given part (inside the Edje theme file).

The BOX layout can be used through the `elm_layout_box_*()` set of functions. It is very similar to the `elm_box` widget but the BOX layout's behavior is completely controlled by the Edje theme. The TABLE layout is like the BOX layout, the difference is that it is used through the `elm_layout_table_*()` set of functions.

Signals

Elm can send Edje signals to the EDC part by using the `elm_layout_signal_emit` . You can also use `elm_layout_signal_callback_add` to receive signals.

Use the following code to listen to any signals sent by the layout:

```
elm_layout_signal_callback_add(layout, "*", "*", _signal_cb, NULL);

static void
_signal_cb(void *data, Evas_Object *obj, const char *emission, const char *source)
{
   printf (http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Info re
ceived from layout : %s %s\n", emission, source);
}
```

For more details on this, see the section on edje signals and layouts (/develop/legacy/program_guide/event_effect/edje_events#Usual_Usage_for_Parts_Aggregated_in_Groupswith_Layouts).

Edje Blocks

A typical Edje file starts with the following definitions:

```
images {}
fonts {}
color_classes {}
styles {}
collections
{
   sounds
   {
      tone: "tone-1" 2300;
      tone: "tone-2" 2300;
      sample
      {
         name: "sound1" RAW;
         source: "sound_file1.wav";
      }
   }
   group {}
   group {}
}
```

This guide is based on original work from Andres Blanc (dresb) andresblanc@gmail.com, it has been heavily expanded and edited since then.

Table of Contents

Images Block

```
images
{
   image: "filename1.ext" COMP;
   image: "filename2.ext" LOSSY 99;
   set {}
   set {}
}
```

The `images` block is used to list each image file which are used in the theme. If any compression method is used, it is also defined here. Besides the document's root, additional `images` blocks can be included inside other blocks, usually `collections`, `group`, and `part`. This makes file list maintenance easier when the theme is split among multiple files.

- `RAW` : Uncompressed
- `COMP` : Lossless compression
- `LOSSY [0-100]` : JPEG lossy compression with quality from 0 to 100
- `USER` : Not embedded to the file, refer to the external file instead

Images.set Block

```
set
{
   name: "image_name_used";
   image {}
   image {}
}
```

The `set` block is used to define an image with different content depending on their size. Besides the document's root, additional `set` blocks can be included inside other blocks, normally `collections`, `group` and `part`. This makes file list maintenance easier when the theme is split among multiple files.

- `name [image name]`

Images.set.image Block

```
image
{
   image: "filename4.ext" COMP;
   size: 51 51 200 200;
   border: 0 0 0 0;
   border_scale_by: 0.0;
}
```

The `image` block inside a `set` block defines the characteristic of an image. Every block describes one image and the size rule to use it.

For full documentation, see Image.

- `image [image file] [compression method] (compression level)`

This is included in each image file. The full path to the directory holding the images can be defined later with `edje_cc`'s `-id` option. Available compression methods are:
    - `RAW` : Uncompressed
    - `COMP` : Lossless compression
    - `LOSSY [0-100]` : JPEG lossy compression with quality from 0 to 100
    - `USER` : Not embedded to the file, refer to the external file instead

- `size [minw] [minh] [maxw] [maxh]`
  Defines the minimum and maximum size that selects the specified image.

- `border [left] [right] [top] [bottom]`
  If set, the area (in pixels) of each side of the image is displayed as a fixed size border, from the side to inwards, preventing the corners from being changed on a resize.

- `border_scale_by [value]`
  If border scaling is enabled then normally the OUTPUT border sizes (e.g. if 3 pixels on the left edge are set as a border, then normally at scale 1.0, those 3 columns are always the exact 3 columns of output, or at scale 2.0 they will be 6 columns, or 0.33 they will merge into a single column). This property multiplies the input scale factor by this multiplier, allowing the creation of supersampled borders to make much higher resolution outputs possible by always using the highest resolution artwork and then runtime scaling it down. Valid values are: 0.0 or bigger (0.0 or 1.0 to turn it off).

Fonts Block

```
fonts
{
   font: "filename1.ext" "fontname";
   font: "filename2.ext" "otherfontname";
}
```

The `fonts` block is used to list each font file with an alias used later in the theme. As with the `images` block, additional `fonts` blocks can be included inside other blocks.

- `font [font filename] [font alias]`
  Defines each `font file` and `alias`, the full path to the directory containing the font files can be defined with `edje_cc`'s `-fd` option.

Data Block

```
data
{
   item: "key" "value";
   file: "otherkey" "filename.ext";
}
```

This block behaves in the same way as the data block inside the parts, only the use-cases are different: this is useful for data that is for the whole theme, for instance license information, version, authors, and so on.

Color_classes Block

```
color_classes
{
   color_class
   {
      name: "colorclassname";
      color: [0-255] [0-255] [0-255] [0-255];
      color2: [0-255] [0-255] [0-255] [0-255];
      color3: [0-255] [0-255] [0-255] [0-255];
   }
}
```

The `color_classes` block contains a list of one or more `color_class` blocks. Each `color_class` allows the designer to name an arbitrary group of colors to be used in the theme, the application can use that name to alter the color values at runtime.

- `name [color class name]`
  Sets the name for the color class, used as reference by both the theme and

the application.

- `color [red] [green] [blue] [alpha]`
  The main color.
- `color2 [red] [green] [blue] [alpha]`
  Used as outline in text and textblock parts.
- `color3 [red] [green] [blue] [alpha]`
  Used as shadow in text and textblock parts.

Styles Block

```
styles
{
   style
   {
      name: "stylename";
      base: "..default style properties..";
      tag: "tagname" "..style properties..";
   }
}
```

The `styles` block contains a list of one or more `style` blocks. A `style` block is used to create style `<tags>` for advanced TEXTBLOCK formatting.

- `name [style name]`

The name of the style to be used as reference later in the theme.

- `base [style properties string]` The default style properties that is applied to the complete text. The available tags that can be used in the style property are the following:
  - font
  - font_size
  - color
  - color_class
  - text_class
  - ellipsis
  - wrap
  - style
  - valign
  - align

    The font size is a point size, and the size of the rendered text is affected by the ppi information. The system basic ppi is 96, and you can change it on the Emulator menu.

- `tag [tag name] [style properties string]`
  Style to be applied only to text between style `<tags>..</tags>` . When creating paired tags, like `<bold></bold>` , a '+' sign must be added at the start of the style properties of the first part (`<bold>`). If the second part (`</bold>`) is also defined, a '-' sign must be added to its style properties. This applies only to paired tags; single tags, like `<tab>` , must not have a starting '+'.
  - br
  - tab
  - b
  - match

The following code is the sample of a text style for the text style:

```
style
{
   name: "list_text_main";
   base: "font=Sans:style=Regular font_size=30 color=#ffffff ellipsis=1.0";
   tag: "br" "\n";\
   tag: "ps" "ps";\
   tag: "tab" "\t";\
   tag: "b" "+ font_weight=Bold";
}
```

Collections Block

```
collections
{
   base_scale: 1.8;
   sounds
   {
      tone: "tone-1" 2300;
      tone: "tone-2" 440;
      sample
      {
         name: "sound_file1" RAW;
         source: "sound_file1.wav";
      }
   }
   group
   {
      name: "groupname";
      alias: "anothername;"
      min: width height;
      max: width height;
      parts {}
      scripts {}
      limits {}
      data {}
      programs {}
   }
}
```

The `collections` block is used to list the groups that compose the theme. Additional `collections` blocks do not prevent overriding group names.

The `sounds` block contains a list of one or more sound sample and tone items.

Base Scale

- `base_scale: [scale factor]`
  Information about the scale factor in the edc file.

Object scaling must be defined in the `config.xml` file of this application to show the application in a proper size in other devices. However, the scaling must be based on the scale 1.0 or, if the application is based on another scale, this scale must be defined in the `config.xml` file. This predefined scale is called the “base scale”.

The size of a scalable object is multiplied with the device scale value. If the scalable object with the size 10 is created in a device with the scale 1.0, the size of the object is 20 in a device with the scale 2.0, and 40 in a device with the scale 4.0.

Sounds Block

- `tone [tone name] [frequency]`
  A sound of the given frequency.

Sounds.sample Block

The sample block defines the sound sample.

- `name [sample name] [compression type] (quality)`
  Used to include each sound file. The full path to the directory holding the sounds can be defined later with `edje_cc`'s `-sd` option. Valid types are:
  - `RAW` : Uncompressed
  - `COMP` : Lossless compression
  - `LOSSY [-0.1 - 1.0]` : Lossy compression with quality from 0.0 to 1.0
  - `AS_IS` : No compression or encoding, write the file information as it is
- `source [file name]`
  The sound source file name (source can be mono/stereo WAV file. Only files with 44.1 KHz sample rate are supported for now).

Group Block

```
group
{
   name: "nameusedbytheapplication";
   alias: "anothername";
   min: width height;
   max: width height;
   parts
   {
      <definitions of parts>
   }
   script
   {
      <embryo script>
   }
   limits
   {
      vertical: "limit_name" height_barrier;
      horizontal: "limit_name" width_barrier;
   }
   data
   {
      items: "key" "value";
      file: "key" "file";
   }
   programs {}
}
```

Table of Contents

Group

A `group` block contains the list of parts and programs that compose a given Edje Object.

- `name [group name]`

The name that is used by the application to load the resulting Edje object and to identify the group to swallow in a GROUP part. If a group with the same name exists already it will be completely overridden by the new group.

- `inherit_only [1 or 0]`
  This flags the group as being used only for inheriting, which will inhibit `edje_cc` resolving of programs and parts that may not exist in this group, but are located in the group which is inheriting this group.
- `inherit [parent group name]`
  Parent group name for inheritance. Group `inherit` is used to inherit any predefined group and change some property which belongs to `part` , `description` , `items` or `program` . The child group has the same properties as its parent group. If you specify the type again in an inherited part, it causes an error.

  When inheriting any parts, descriptions without state names are not allowed.

- `script_recursion [1/0]`
  This flag (1 or 0) determines whether to error on unsafe calls when recursively running Embryo programs. For example, running an Embryo script which calls EDC that has a `script{}` block is unsafe, and the outer-most (first) Embryo stack is corrupted. It is strongly unadvisable to use this flag.
- `alias [additional group name]`
  Additional name to be used as an identifier. Defining multiple aliases is supported.
- `min [width] [height]`
  The minimum size for the container defined by the composition of the parts. It is not enforced.
- `max [width] [height]`
  The maximum size for the container defined by the totality of the parts. It is not enforced.
- `broadcast_signal [on/off]`
  Signal gets automatically broadcasted to all sub group parts. Default is “true”.
- `orientation [AUTO/LTR/RTL]`
  This defines GROUP orientation. It is useful if you want match interface orientation with language.
  - `AUTO` : Follow system defaults
  - `LTR` : Used in Left To Right Languages (Latin)
  - `RTL` : Used in Right To Left Languages (Hebrew, Arabic interface)
- `mouse_events [1 or 0]`
  Changes the default value of `mouse_events` for every part in this group. Default is “1”, to maintain compatibility.
- `program_source [source name]`
  Change the default value of source for every program in the current group which is declared after this value is set. Defaults to an unset value to maintain compatibility. The name of source can be set on every program, but if the name can be defined in the group level, it reduces the effort to indicate it in every program.

Group.parts

```
parts
{
   part {}
   part {}
   part {}
}
```

The `parts` group contain one or more `part`. Each part describes a visual element and has a type: `text`, `image`, `table`, etc.

For complete documentation, see the Part Block guide.

Group.script

```
group
{
   script
   {
      <embryo script>
   }
   program
   {
      script
      {
         <embryo script>
      }
   }
}
```

This block is used to inject embryo scripts to a given Edje theme and it functions in two modalities. When it is included inside a `program` block, the script is executed every time the program is run, on the other hand, when included directly into a `group`, `part`, or `description` block, it is executed once at the load time, in the load order.

For more information, see the Program guide.

Group.limits

```
limits
{
   vertical: "limit_name" height_barrier;
   horizontal: "limit_name" width_barrier;
}
```

This block is used to trigger signals when the Edje object is resized.

- `vertical [name] [height barrier]`
  This sends a signal `limit,name,over` when the object is resized and passes the limit by growing over it. When the object's size is reduced below the limit, signal `limit,name,below` is sent. This limit is applied on the y axis and is given in pixels.
- `horizontal [name] [width barrier]`
  This sends a signal `limit,name,over` when the object is resized and passes the limit by growing over it. When the object's size is reduced below the limit, signal `limit,name,below` is sent. This limit is applied on the x axis and is given in pixels.

Group.data

```
data
{
   item: "key" "value";
   file: "key2" "somefile";
}
```

The `data` block is used to pass arbitrary parameters from the theme to the application. Unlike the `images` and `fonts` blocks, additional `data` blocks can only be included inside the `group` block.

- `items: “key” “value”;`
  Defines a new parameter, the value is the string specified next to it.
- `file: “key” “file”;`
  Defines a new parameter, the value is the contents of the specified file formatted as a single string of text. This property only works with plain text files.

> For genlist item styles, keys must be texts and icons, respectively for text parts and swallow parts; values must the names of the parts, separated with spaces.

Group.programs

```
programs
{
   program {}
   program {}
   program {}
}
```

The `programs` group contain one or more `program` .

For more information about the program structure, see the Program guide.

Part Block

Table of Contents

```
part
{
   // inherit all the fields of another part
   inherit: "partname";
   // name the part
   name: "partname";
   // set the part type
   type: IMAGE;
   // enable mouse events on the part
   mouse_events: 0/1;
   // repeat mouse events to parts below the current one
   repeat_events: 0/1;
   ignore_flags: NONE;
   // whether the part sizes scale with the edje scaling factor
   scale: 0/1;
   // whether fully-transparent pixels are taken into account for collision detection
   precise_is_inside: 0/1;
   // only render the area of the part that coincides with the given part
   clip_to: "anotherpart";
   // for group/textblock parts: swallow the given group (for textblock: goes below tex
t)
   source: "groupname";
   // same as source but goes on top of text
   source2: "groupname";
   // for textblock parts: swallow the given group below the mouse cursor when it hovers
over the part
   source3: "groupname";
   // same as source3 but goes on top of the cursor
   source4: "groupname";
   // for textblock parts: swallow the given group below text anchors (&lt;a&gt;...&lt;/
a&gt;)
   source5: "groupname";
   // same as source5 but goes on to top of the anchor
   source6: "groupname";
   // add a shadow effect to the part
   effect: SOFT_SHADOW (BOTTOM_RIGHT);
   // for textblock parts: makes the text editable, possibly with a specific behavior
   entry_mode: PASSWORD;
   // for textblock parts: change how the tex selection is triggered
   select_mode: EXPLICIT;
   // for editable textblock parts: where to draw the (blinking) cursor
   cursor_mode: UNDER;
   // for editable textblock parts: allow multiple lines
   multiline: 0/1;
   // for textblock parts: accessibility features will be used
   access: 0/1;
   // no-one uses that
   pointer_mode: AUTOGRAB;
   use_alternate_font_metrics: 0/1;
   // remove the given program; useful when it was inherited
   program_remove: "programname";
```

```
   // remove the given part; useful when it was inherited
   part_remove: "partname";
   // insert the current part below the given part, as if it were declared before
   insert_before: "partname";
   // insert the current part above the given part, as if it were declared after
   insert_after: "partname";
   // define a new part inside this one
   part
   {
      <part definition>
   }
   draggable
   {
      // confine the current part to the given part
      confine: "another part";
      // only start drag when it would have moved enough to be outside of the given part
      threshold: "another part";
      // forward drag events to the given part instead of handling them
      events: "another draggable part";
      // enable horizontal drag of the part with steps each step_size or with steps_coun
t steps
      x: 0/1 &lt;step_size&gt; &lt;steps_count&gt;;
      // same as x but vertical
      y: 1 0 100;
   }
   // for box or table parts
   box/table
   {
      // list of items
      items
      {
         item
         {
            // define item type, can only be GROUP
            type: GROUP;
            // name the item
            name: "name";
            // set the source for this item, i.e. its contents
            source: "groupname";
            // minimum horizontal and vertical item sizes (-1 for expand)
            min: -1 -1;
            // maximum horizontal and vertical item sizes (-1 for ignore)
            max: 100 100;
            // set the item padding in pixels
            padding: 2 2 2 2;
            spread: 1 1;
            // set the item alignment
            align: 0.5 0.5;
            // set a weight hint in the box for the given object
            weight: 1 1;
            // set the aspect ratio hint
            aspect: 1 1;
```

```
            aspect_mode: BOTH;
            // number of columns and rows the item will take
            span: 1 1;
         }
      }
   }
   description
   {
      // inherit another description
      inherit: "default" 1.0;
      // name the current description
      state: "default" 1.0;
      // use another part as content of the current "PROXY" part (This description only
works in the PROXY part and the current part mirrors the rendering content of the source
part)
      source: "partname";
      // make the part (in)visible (invisible parts emit no signals)
      visible: 0/1;
      // emit a signal when the given dimension becomes zero or stops being zero
      limit: WIDTH;
      // horizontal and vertical alignment of the part inside its parent
      align: 0.5 0.5;
      // set that the part does not change size
      fixed: 0/1 0/1;
      // set the minimum size for the part
      min: 200 200;
      // forcibly multiply the minimum sizes by the given factors
      minmul: 1.2 1.2;
      // set the maximum size for the part
      max: 400 400;
      // make vertical and horizontal resizes happen in steps
      step: 0 0;
      // force aspect ratio to be kept between min and max between resizes
      aspect: 0.8 1.2;
      // the dimension to which the aspect ratio applies
      aspect_preference: BOTH;
      // use the given color class which can be used to factor font colors
      color_class: "colorclassname";
      // set the text color
      color: 255 0 0 255;
      // set the color of the text's shadow
      color2: 0 255 0 255;
      // set the color of the text's outline
      color3: 0 0 255 255;
      // define the positions of the top-left (rel1) and bottom-right (rel2) corners
      rel1/rel2
      {
         // make relative and offset use the given part for their positioning
         to: "partname";
         // position the corner relative to the part given through to (0.0 being axis
ginning, 1.0 being its end)
         relative: 0.1 0.1;
```

```
         // add an absolute offset (in pixels) along each axis
         offset: 10 10;
         // same as to but only for the x axis
         to_x: "partname";
         // same as to but only for the y axis
         to_y: "partname";
      }
      // settings specific to parts of type image
      image
      {
         // name (not path) of the regular image
         normal: "imagename";
         // image to use while transitioning to the normal image; use several times to c
reate animations
         tween: "imagename2";
         // set the number of pixels that make up each border of the image, i.e. are not
resized when the image is
         border: 4 4 4 4;
         // hide, strip from its alpha or show (default) the non-border part of the imag
e
         middle: DEFAULT;
         border_scale_by: 1.0;
         // whether to scale the border or not
         border_scale: 0/1;
         scale_hint: STATIC;
         // set how the image is going to fill its part
         fill:
         {
            // whether to smooth the image when scaling it
            smooth: 0/1;
            spread: ??;
            // whether to scale or tile to fit when resizing the image is needed
            type: SCALE/TILE;
            // only display the part of the image that is below and to the right of the
given point
            origin
            {
               // specify top-left point as relative coordinates
               relative: 0.1 0.1;
               // specify top-left point as a pixel offset
               offset: 10 10;
            }
            // specify bottom-right point as relative coordinates
            size
            {
               relative: 0.1 0.1;
               // specify bottom-right point as a pixel offset
               offset: 10 10;
            }
         }
      }
      // settings specific to parts of type text and textblock
```

```
      text
      {
         // set the part's text
         text: "some text";
         // set the text's font
         font: "Sans";
         // set the text's size
         size: 14;
         // set the text's class which can be used to factor font and font-size settings
         text_class: "classname";
         // use the styles defined in stylename
         style: "stylename";
         // for textblocks in password mode, replace characters to hide with this string
         repch: "*";
         // set the min and max font sizes allowed when resizing (default is 0 0, i.e. u
nrestricted)
         size_range: 6 18;
         // increase font size as much as possible while still remaining in the containe
r for both axis
         fit: 0/1 0/1;
         // make the min size of the container equal to the min size of the current text
(0 0 by default)
         min: 0/1 0/1;
         // make the max size of the container equal to the max size of the current text
(0 0 by default)
         max: 0/1 0/1;
         // set the vertical and horizontal alignments of the text
         align: 0.5 0.5;
         // re-use the text of another part
         text_source: "partname";
         // when text is too long to fit, relative position at which to cut the text and
put an ellipsis ("...")
         ellipsis: 0.9;
      }
      // settings specific to parts of type box
      box
      {
         // set how children while be arranged in the box
         layout: horizontal_homogeneous;
         // set the vertical and horizontal alignments of box
         align: 0.5 0.5;
         // set the padding between items of the box
         padding: 1 1;
         // make the box' min size be the min size of its elements (i.e. make it shrinka
ble as much as its items)
         min: 0 1;
      }
      table
      {
         // make items homogeneous
         homogeneous: NONE;
         // set the vertical and horizontal alignments of box
```

```
         align: 0.5 0.5;
         // set the padding between items of the box
         padding: 1 1;
         // make the table's min size be the min size of its elements (i.e. make it shri
nkable as much as its items)
         min: 0 1;
      }
      map
      {
         perspective: "partname";
         light: "partname";
         on: 0/1;
         smooth: 0/1;
         alpha: 0/1;
         backface_cull: 0/1;
         perspective_on: 0/1;
         color: 0/1;
         rotation
         {
            center: "partname";
            x: 45;
            y: 120;
            z: 90;
         }
         perspective
         {
            zplane: 0/1;
            focal: 20;
         }
         // simpler syntax to create transitions to the current part
         link
         {
            base: "edje,signal" "edje";
         }
      }
   }
}
```

Part

```
part
{
   name: "partname";
   type: IMAGE;
   mouse_events:  1;
   repeat_events: 0;
   ignore_flags: NONE;
   clip_to: "anotherpart";
   source:  "groupname";
   pointer_mode: AUTOGRAB;
   use_alternate_font_metrics: 0;

   dragable {}
   items {}
   description {}
}
```

Parts are used to represent the most basic design elements of the theme, for example, a part can represent a line in a border or a label on a button.

- `inherit [part name]`
  Copies all attributes except part name from referenced part into current part. All existing attributes, except part name, are overwritten.

  > When inheriting any parts, descriptions without state names are not allowed.

- `program_remove [program name] (program name) (program name) …`
  Removes the listed programs from an inherited group. Removing non-existing programs is not allowed.

  > This breaks program sequences if a program in the middle of the sequence is removed.

- `part_remove [part name] (part name) (part name) …`
  Removes the listed parts from an inherited group. Removing non-existing parts is not allowed.
- `name [part name]`
  The part's name is used as reference in the theme's relative positioning system, by programs and in some cases by the application. It must be unique within the group.
- `type [TYPE]`
  Sets the type. This is set to IMAGE by default. Valid types are:
  - `RECT` : Rectangle object in the screen
  - `TEXT` : Simple text
  - `IMAGE` : Image area
  - `SWALLOW` : Area where you can add the object
  - `TEXTBLOCK` : Complex text with multiple lines, mark-up elements, and such.

    - `GROUP` : Part which can include other groups in the same group.
    - `BOX` : Container object as a container. It has a row or column.
    - `TABLE` : Container object as a container. It has a row and column.
    - `PROXY` : Clone of another part in the same group. It shares the memory of the source part.
    - `SPACER` : Rectangle object, but invisible. Recommended to padding because it does not allocate any memory.

Nested parts adds hierarchy to Edje. Nested part inherits its location relatively to the parent part. To declare a nested part create a new part within current part declaration. Define parent part name before adding nested parts.

```
part
{
   name: "parent_rect";
   type: RECT;
   description {}
   part
   {
      name: "nested_rect";
      type: RECT;
      description {}
   }
}
```

- `insert_before [another part's name]`
  The part's name which this part is inserted before. One part cannot have both `insert_before` and `insert_after` . One part cannot refer more than one by `insert_before` .
- `insert_after [another part's name]`
  The part's name which this part is inserted after. One part cannot have both `insert_before` and `insert_after` . One part cannot refer more than one by `insert_after` .
- `mouse_events [1 or 0]`
  Specifies whether the part emits signals, although it is named `mouse_events` . Disabling it (0) prevents the part from emitting signal. It is set to 1 by default, or to the value set to `mouse_events` at the group level, if any.
- `repeat_events [1 or 0]`
  Specifies whether a part echoes a mouse event to other parts below the pointer (1), or not (0). It is set to 0 by default.
- `ignore_flags [FLAG] …` Specifies whether events with the given flags are ignored, i.e., do not emit signals to the parts. Multiple flags must be separated by spaces, the effect is ignoring all events with one of the flags specified. Possible flags are:
    - `NONE` : Event is handled properly (default value)
    - `ON_HOLD` : Event is not handled or passed in this part
- `scale [1 or 0]`
  Specifies whether the part scales its size with an Edje scaling factor. By default scale is off (0) and the default scale factor is 1.0 which means no scaling. This is used to scale properties such as font size, min/

max size of the part, and it can also be used to scale based on DPI of the target device. The reason to be selective is that some parts are scaled well, others are not, so choose what works best.

- `pointer_mode [MODE]`
  Sets the mouse pointer behavior for a given part. The default value is AUTOGRAB. Available modes are:
  - `AUTOGRAB` , when the part is clicked and the button remains pressed, the part is the source of all future mouse signals emitted, even outside the object, until the button is released.
  - `NOGRAB` , the effect is limited to the part's container.

- `precise_is_inside [1 or 0]`
  Enables precise point collision detection for the part, which is more resource-intensive. It is disabled by default.

- `use_alternate_font_metrics [1 or 0]`
  Only affects text and textblock parts, when enabled Edje uses different size measurement functions. It is disabled by default.

- `clip_to [another part's name]`
  Only renders the area of part that coincides with another part's container. Overflowing content is not displayed. Note that the part being clipped to can only be a rectangle part.

- `source [another group's name]`
  Only available to GROUP or TEXTBLOCK parts. Swallows the specified group into the part's container if it is a GROUP. If TEXTBLOCK it is used for the group to be loaded and used for selection display UNDER the selected text. `source2` is used for on top of the selected text, if `source2` is specified.

- `source2 [another group's name]`
  Only available to TEXTBLOCK parts. It is used for the group to be loaded and used for selection display OVER the selected text. `source` is used for under of the selected text, if `source` is specified.

- `source3 [another group's name]`
  Only available to TEXTBLOCK parts. It is used for the group to be loaded and used for cursor display UNDER the cursor position. `source4` is used for over the cursor text, if `source4` is specified.

- `source4 [another group's name]`
  Only available to TEXTBLOCK parts. It is used for the group to be loaded and used for cursor display OVER the cursor position. `source3` is used for under the cursor text, if `source4` is specified.

- `source5 [another group's name]`
  Only available to TEXTBLOCK parts. It is used for the group to be loaded and used for anchors display UNDER the anchor position. `source6` is used for over the anchors text, if `source6` is specified.

- `source6 [another group's name]`
  Only available to TEXTBLOCK parts. It is used for the group to be loaded and used for anchor display OVER the anchor position. `source5` is used for under the anchor text, if `source6` is specified.

- `effect [effect]` (shadow direction)
  Apply the selected outline, shadow, or glow effect to “textblock” (take care that this effect only works for the textblock). The available effects are:
  - `PLAIN`
  - `OUTLINE`

- SOFT_OUTLINE
- SHADOW
- SOFT_SHADOW
- OUTLINE_SHADOW
- OUTLINE_SOFT_SHADOW
- FAR_SHADOW
- FAR_SOFT_SHADOW
- GLOW

The available shadow directions definitions are (default is BOTTOM_RIGHT):

- BOTTOM_RIGHT
- BOTTOM
- BOTTOM_LEFT
- LEFT
- TOP_LEFT
- TOP
- TOP_RIGHT
- RIGHT

- entry_mode [mode]
  Sets the edit mode for a textblock part. The available modes are:
  - NONE :
    The textblock is non-editable.
  - PLAIN :
    The textblock is non-editable, but selectable.
  - EDITABLE :
    The textblock is editable.
  - PASSWORD :
    The textblock is editable if the Edje object has the keyboard focus and the part has the Edje focus (or selectable always regardless of focus). In the event of password mode, not selectable and all text chars replaced with *'s but editable and pastable.

- select_mode [mode]
  Sets the selection mode for a textblock part. The available modes are:
  - DEFAULT , selection mode is what you would expect on any desktop. Press mouse, drag and release to end.
  - EXPLICIT , this mode requires the application controlling the Edje object has to explicitly begin and end selection modes, and the selection itself is draggable at both ends.

- cursor_mode [mode]
  Sets the cursor mode for a textblock part. The available modes are:
  - UNDER , the cursor draws below the character pointed at. That is the default.
  - BEFORE , the cursor is drawn as a vertical line before the current character, just like many other GUI (Graphical User Interface) toolkits handle it.

- multiline [1 or 0]
  It causes a textblock that is editable to allow multiple lines for editing.

- `access [1 or 0]`
  Specifies whether the part uses accessibility feature (1), or not (0). It is set to 0 by default.

Draggable

```
dragable
{
   confine: "another part";
   threshold: "another part";
   events:  "another draggable part";
   x: 0 0 0;
   y: 0 0 0;
}
```

When this block is used the part can be dragged around the interface, do not confuse with external drag and drop. By default Edje (and most applications) attempts to use the minimal size possible for a draggable part. If the min property is not set in the description the part is (most likely) set to 0px width and 0px height, thus invisible.

- `x [enable/disable] [step] [count]`
  Used to set up dragging events for the X axis. The first parameter is used to enable (1 or -1) and disable (0) dragging along the axis. When enabled, 1 sets the starting point at 0.0 and -1 at 1.0. The second parameter takes any integer and limits movement to values divisible by it, causing the part to jump from position to position. If step is set to 0 it is calculated as width of confine part divided by count.
- `y [enable/disable] [step] [count]`
  Used to set up dragging events for the Y axis. The first parameter is used to enable (1 or -1) and disable (0) dragging along the axis. When enabled, 1 sets the starting point at 0.0 and -1 at 1.0. The second parameter takes any integer and limits movement to values divisible by it, causing the part to jump from position to position. If step is set to 0 it is calculated as height of confine part divided by count.
- `confine [another part's name]`
  Limits the movement of the dragged part to another part's container. Set a min size for the part, or the dragged object will not show up.
- `threshold [another part's name]`
  When set, the movement of the dragged part can only start when it get moved enough to be outside of the threshold part.
- `events [another draggable part's name]`
  Causes the part to forward the drag events to another part, thus ignoring them for itself.

Box/table

Items

```
box/table
{
   items
   {
      item
      {
         type: GROUP;
         source: "some source";
         min: -1 -1;
         max: 100 100;
         padding: 1 1 2 2;
      }
      item
      {
         type: GROUP;
         source: "some other source";
         name: "some name";
         align: 1.0 0.5;
      }
   }
}
```

On a BOX part, this block is used to set other groups as elements of the box. These can be mixed with external objects set by the application through the `edje_object_part_box*` API (Application Programming Interface).

Item

- `type [item type]`
  Sets the type of the object this item holds. The supported type is:
  - `GROUP` (default)
- `name [name for the object]`
  Sets the name of the object via `evas_object_name_set()`.
- `source [another group's name]`
  Sets the group this object is made of.
- `min [width] [height]`
  Sets the minimum size hints for this object.

  Must be -1 to get expand behavior.
- `spread [width] [height]`
  Replicates the item in a rectangle of size width x height box starting from the defined position of this item. The default value is 1 1;.
- `prefer [width] [height]`
  Sets the preferred size hints for this object.

- `max [width] [height]`
  Sets the maximum size hints for this object.
- `padding [left] [right] [top] [bottom]`
  Sets the padding hints for this object.
- `align [x] [y]`
  Sets the alignment hints for this object.
- `weight [x] [y]`
  Sets the weight hints for this object.
- `aspect [w] [h]`
  Sets the aspect width and height hints for this object.
- `aspect_mode [mode]`
  Sets the aspect control hints for this object. The available hints are:
  - `NONE`
  - `NEITHER`
  - `HORIZONTAL`
  - `VERTICAL`
  - `BOTH`
- `options [extra options]`
  Sets extra options for the object.
- `position [col] [row]`
  Sets the position this item has in the table. This is required for parts of type TABLE.
- `span [col] [row]`
  Sets how many columns and rows this item uses. The default value is 1 1.

Description

```
description
{
   inherit: "another_description" INDEX;
   state: "description_name" INDEX;
   visible: 1;
   min: 0 0;
   max: -1 -1;
   align: 0.5 0.5;
   fixed: 0 0;
   step: 0 0;
   aspect: 1 1;

   rel1
   {
      ...
   }

   rel2
   {
      ...
   }
}
```

Every part can have one or more description blocks. Each description is used to define style and layout properties of a part in a given “state”.

- `inherit [another description's name] [another description's index]`
  Thee description inherits all the properties from the named description. The properties defined in this part override the inherited properties, reducing the amount of necessary code for simple state changes. Note: inheritance in Edje is single level only.
- `source [another part's name]`
  Causes the part to use another part's content as the content of this part. This works only with PROXY part.
- `state [name for the description] [index]`
  Sets a name used to identify a description inside a given part. Multiple descriptions are used to declare different states of the same part, like “clicked” or “invisible”. All state declarations are also coupled with an index number between 0.0 and 1.0. All parts must have at least one description named “default 0.0”.
- `visible [0 or 1]`
  Takes a Boolean value specifying whether part is visible (1) or not (0). Non-visible parts do not emit signals. The default value is 1.
- `limit [mode]`
  Emits a signal when the part size changes from zero or to a zero (limit,width,over, limit,width,zero). By default no signal are emitted. Valid values are:
  - `NONE`
  - `WIDTH`
  - `HEIGHT`

    - BOTH

- `align [X axis] [Y axis]`
  When the displayed object's size is smaller or bigger than its container, this property moves it relatively along both axis inside its container. “0.0” means left/top edges of the object touching container's respective ones, and “1.0” stands for right/bottom edges of the object (on horizontal/vertical axis, respectively). The default value is “0.5 0.5”.

- `fixed [width, 0 or 1] [height, 0 or 1]`
  Sets the minimum size calculation. See `edje_object_size_min_calc()` and `edje_object_size_min_restricted_calc()` . This tells the min size calculation routine that this part does not change size in width or height (1 for it does not, 0 for it does), so the routine does not try to expand or contract the part.

- `min [width] [height] or SOURCE`
  Sets the minimum size of the state. When min is defined to SOURCE, it looks at the original image size and enforces its minimal size to match at least the original one. The part must be an IMAGE or a GROUP part.

- `minmul [width multiplier] [height multiplier]`
  A multiplier forcibly applied to whatever minimum size is only during minimum size calculation.

- `max [width] [height] or SOURCE`
  The maximum size of the state. A size of -1.0 means that it is ignored in one direction. When max is set to SOURCE, Edje enforces the part to be not more than the original image size. The part must be an IMAGE part.

- `step [width] [height]`
  Restricts resizing of each dimension to values divisible by its value. This causes the part to jump from value to value while resizing. The default value is “0 0” which disables stepping.

- `aspect [min] [max]`
  Normally width and height can be resized to any values independently. The aspect property forces the width to height ratio to be kept between the minimum and maximum set. For example, “1.0 1.0” increases the width a pixel for every pixel added to height. The default value is “0.0 0.0” which disables aspect. For a more detailed explanation, see the Min Size Hint .

- `aspect_preference [DIMENSION]`
  Set the dimensions to which the “aspect” property applies. Available options are:
    - BOTH
    - VERTICAL
    - HORIZONTAL
    - SOURCE
    - NONE

- `color_class [color class name]`
  The part uses the color values of the named `color_class` , these values can be overridden by the “color”, “color2” and “color3” properties.

- `color [red] [green] [blue] [alpha]`
  Sets the main color to the specified values (between 0 and 255).

  > The textblock part is not affected by the color description. Set the color in the text style.

- `color2 [red] [green] [blue] [alpha]`
  Sets the text shadow color to the specified values (0 to 255).
- `color3 [red] [green] [blue] [alpha]`
  Sets the text outline color to the specified values (0 to 255).

Rel1/rel2

```
description
{
   rel1
   {
      relative: 0.0 0.0;
      offset:     0   0;
   }
   rel2
   {
      relative: 1.0 1.0;
      offset:    -1  -1;
   }
}
```

The rel1 and rel2 blocks are used to define the position of each corner of the part's container. With rel1 being the left-up corner and rel2 being the right-down corner.

- `relative [X axis] [Y axis]`
  Moves the corner to a relative position inside the container of the relative “to” part. Values from 0.0 (0%, beginning) to 1.0 (100%, end) of each axis.
- `offset [X axis] [Y axis]`
  Affects the corner position a fixed number of pixels along each axis.
- `to [another part's name]`
  Positions the corner relatively to another part's container. Setting to “” resets this value for inherited parts.
- `to_x [another part's name]`
  Positions the corner relatively to the X axis of another part's container. This affects the first parameter of “relative”. Setting to “” resets this value for inherited parts.
- `to_y [another part's name]`
  Positions the corner relatively to the Y axis of another part's container. This affects the second para of “relative”. Setting to “” resets this value for inherited parts.

Image

```
description
{
   image
   {
      normal: "filename.ext";
      tween:  "filename2.ext";
      tween:  "filenameN.ext";
      border:  left right top bottom;
      middle:  0/1/NONE/DEFAULT/SOLID;
      fill {}
   }
}
```

- `normal [image's filename]`
  Name of image to be used as previously declared in the images block. In an animation, this is the first and last image displayed. It is required in any image part
- `tween [image's filename]`
  Name of an image to be used in an animation loop, an image block can have none, one or multiple tween declarations. Images are displayed in the order they are listed, during the transition to the state they are declared in; the “normal” image is the final state.
- `border [left] [right] [top] [bottom]`
  Sets the area (in pixels) of each side of the image is displayed as a fixed size border, from the side -> inwards, preventing the corners from being changed on a resize.
- `middle [mode]`
  If border is set, this value tells Edje if the rest of the image (not covered by the defined border) displayed or not or be assumed to be solid (without alpha). The default value is `1/DEFAULT` . The available values are:
  - `0` or `NONE`
  - `1` or `DEFAULT`
  - `SOLID` (strip alpha from the image over the middle zone)
- `border_scale_by [value]`
  If border scaling is enabled then normally the OUTPUT border sizes (e.g. if 3 pixels on the left edge are set as a border, then normally at scale 1.0, those 3 columns are always exactly 3 columns of output, or at scale 2.0 they are 6 columns, or 0.33 they merge into a single column). This property multiplies the input scale factor by this multiplier, allowing the creation of supersampled borders to make higher resolution outputs possible by always using the highest resolution artwork and then runtime scaling it down. Value can be: 0.0 or bigger (0.0 or 1.0 to turn it off)
- `border_scale [0/1]`
  Tells Edje if the border is scaled by the object/global Edje scale factors.

- `scale_hint [mode]`
  Sets the evas image scale hint letting the engine more effectively save cached copies of the scaled image if it makes sense. Valid values are:
    - `0` or `NONE`
    - `DYNAMIC`
    - `STATIC`

Image.fill

```
image
{
   fill
   {
      type: SCALE;
      smooth: 0-1;
      origin {}
      size {}
   }
}
```

The fill method is an optional block that defines the way an IMAGE part is going to be displayed inside its container. It can be used for tiling (repeating the image) or displaying only part of an image. See `evas_object_image_fill_set()` documentation for more details.

- `smooth [0 or 1]`
  The smooth property takes a boolean value to decide if the image will be smoothed on scaling (1) or not (0). The default value is 1.
- `spread`
- `type [fill type]`
  Sets the image fill type. The part parameter “min” must be set, it is size of tiled image. If parameter “max” set tiled area has the size accordingly “max” values. SCALE is default type. Valid values are:
    - `SCALE` , image is scaled accordingly the value of the parameters

“relative” and “offset” from “origin” and “size” blocks.

- `TILE` , image is tiled accordingly parameters value “relative” and “offset” from “origin” and “size” blocks.

```
image
{
   fill
   {
      origin
      {
         relative: 0.0 0.0;
         offset: 0 0;
      }
   }
}
```

The origin block is used to place the starting point, inside the displayed element, that is used to render the tile. By default, the origin is set at the element's left-up corner.

- `relative [X axis] [Y axis]`
  Sets the starting point relatively to displayed element's content.
- `offset [X axis] [Y axis]`
  Affects the starting point a fixed number of pixels along each axis.

```
image
{
   fill
   {
      size
      {
         relative: 1.0 1.0;
         offset: -1 -1;
      }
   }
}
```

The size block defines the tile size of the content that are displayed.

- `relative [width] [height]`
  Takes a pair of decimal values that represent the percentage of the original size of the element. For example, “0.5 0.5” represents half the size, while “2.0 2.0” represents the double. The default value is “1.0 1.0”.
- `offset [X axis] [Y axis]`
  Affects the size of the tile a fixed number of pixels along each axis.

```
text
{
   text: "some string of text to display";
   font: "font_name";
   size: SIZE;
   text_class: "class_name";
   fit: horizontal vertical;
   min: horizontal vertical;
   max: horizontal vertical;
   align: X-axis Y-axis;
   source: "part_name";
   text_source: "text_part_name";
   style: "stylename";
}
```

- `text [a string of text, or nothing]`
  Sets the default content of a text part, normally the application is the one changing its value.
- `text_class [text class name]`
  Similar to color_class, this is the name used by the application to alter the font family and size at ru[illegible].

- `font [font alias]`
  This sets the font family to one of the aliases set up in the “fonts” block. Can be overridden by the application.
- `style [the style name]`
  Causes the part to use the default style and tags defined in the “style” block with the specified name.
- `repch [the replacement character string]`
  If this is a textblock and is in PASSWORD mode this string is used to replace every character to hide the details of the entry. Normally * is used, but you can use anything you like.
- `size [font size in points (pt)]`
  Sets the default font size for the text part. Can be overridden by the application.
- `size_range [font min size in points (pt)] [font max size in points (pt)]`
  Sets the allowed font size for the text part. Setting min and max to 0 means that sizing is not restricted. This is also the default value.
- `fit [horizontal] [vertical]`
  When any of the parameters is set to 1 Edje resizes the text for it to fit in its container. Both are disabled by default.
- `min [horizontal] [vertical]`
  When any of the parameters is enabled (1) it forces the minimum size of the container to be equal to the minimum size of the text. The default value is “0 0”.
- `max [horizontal] [vertical]`
  When any of the parameters is enabled (1) it forces the maximum size of the container to be equal to the maximum size of the text. The default value is “0 0”.
- `align [horizontal] [vertical]` Changes the position of the point of balance inside the container. The default value is 0.5 0.5.
- `source [another TEXT part's name]`
  Causes the part to use the text properties (like font and size) of another part and update them as they change.
- `text_source [another TEXT part's name]`
  Causes the part to display the text content of another part and update them as they change.
- `ellipsis [point of balance]`
  Balances the text in a relative point from 0.0 to 1.0, this point is the last section of the string to be cut out in case of a resize that is smaller than the text itself. The default value is 0.0. Set to -1.0 for no ellipsis.

Box

```
box
{
   layout: "vertical";
   padding: 0 2;
   align: 0.5 0.5;
   min: 0 0;
}
```

A box block can contain other objects and display them in different layouts, any of the predefined set, or a custom one, set by the application.

- `layout [primary layout] (fallback layout)`
  Sets the layout for the box:
  - horizontal (default)
  - vertical
  - horizontal_homogeneous
  - vertical_homogeneous
  - horizontal_max (homogeneous to the max sized child)
  - vertical_max
  - horizontal_flow
  - vertical_flow
  - stack
  - `some_other_custom_layout_set_by_the_application`
    Set a custom layout as a fallback. For more information, see `edje_box_layout_register()` . If an unregistered layout is used, it defaults to horizontal.
- `align [horizontal] [vertical]`
  Changes the position of the point of balance inside the container. The default value is 0.5 0.5.
- `padding [horizontal] [vertical]`
  Sets the space between cells in pixels. The default value is 0 0.
- `min [horizontal] [vertical]`
  When any of the parameters is enabled (1) it forces the minimum size of the box to be equal to the minimum size of the items. The default value is 0 0.

Table

```
table
{
   homogeneous: TABLE;
   padding: 0 2;
   align: 0.5 0.5;
   min: 0 0;
}
```

A table block can contain other objects packed in multiple columns and rows, and each item can span across more than one column and/or row.

- `homogeneous [homogeneous mode]`
  Sets the homogeneous mode for the table:
  - `NONE` : default
  - `TABLE` : available space is evenly divided between children (which overflows onto other children if too little space is available)
  - `ITEM` : size of each item is the largest minimal size of all the items
- `align [horizontal] [vertical]`
  Changes the position of the point of balance inside the container. The default value is 0.5 0.5.
- `padding [horizontal] [vertical]`
  Sets the space between cells in pixels. The default value is 0 0.
- `min [horizontal] [vertical]`
  When any of the parameters is enabled (1), it forces the minimum size of the table to be equal to the minimum size of the items. The default value is 0 0.

Map

```
map
{
   perspective: "name";
   light: "name";
   on: 1;
   smooth: 1;
   perspective_on: 1;
   backface_cull: 1;
   alpha: 1;

   rotation
   {
      ...
   }
}
```

- `perspective [another part's name]`
  This sets the part that is used as the perspective point for giving a part a 3D look. The perspective point must have a perspective section that provides zplane and focal properties. The center of this part is used as the focal point, thus size, color and visibility are not relevant, just center point, zplane and focal are used. This also implicitly enables perspective transforms.
- `light [another part's name]`
  This sets the part that is used as the light for calculating the brightness (based on how directly the part's surface is facing the light source point). Like the perspective point part, the center point is used and zplane is used for the z position (0 being the zero-plane where all 2D objects normally live) and positive values being further away into the distance. The light part color is used as the light color (alpha not used for light color). The color2 color is used for the ambient lighting when calculating brightness (alpha also not used).
- `on [1 or 0]`

This enables mapping for the part. Default is 0.

- `smooth [1 or 0]`
  This enables smooth map rendering. This may be linear interpolation, anisotropic filtering or anything the engine decides is smooth. This is a best-effort hint and may not produce precisely the same results in all engines and situations. The default value is 1.
- `alpha [1 or 0]`
  This enables alpha channel when map rendering. The default value is 1.
- `backface_cull [1 or 0]`
  This enables backface culling (when the rotated part that normally faces the camera is facing away after being rotated etc.). This means that the object are hidden when backface is culled.
- `perspective_on [1 or 0]`
  This enables perspective when rotating even without a perspective point object. This uses perspective set for the object itself or for the canvas as a whole as the global perspective with `edje_perspective_set()` and `edje_perspective_global_set()`.
- `color [point] [red] [green] [blue] [alpha]`
  This sets the color of a vertex in the map. Colors are linearly interpolated between vertex points through the map. The default color of a vertex in a map is white solid (255, 255, 255, 255) which means it has no affect on modifying the part pixels. Currently only four points are supported: 0 - Left-Top point of a part. 1 - Right-Top point of a part. 2 - Left-Bottom point of a part. 3 - Right-Bottom point of a part.

Map.rotation

```
rotation
{
   center: "name";
   x: 45.0;
   y: 45.0;
   z: 45.0;
}
```

Rotates the part, optionally with the center on another part.

- `center [another part's name]`
  This sets the part that is used as the center of rotation when rotating the part with this description. The part's center point is used as the rotation center when applying rotation around the x, y and z axes. If no center is given, the parts original center itself is used for the rotation center.
- `x [X degrees]`
  This sets the rotation around the x axis of the part considering the center set. The value is given in degrees.
- `y [Y degrees]`
  This sets the rotation around the y axis of the part considering the center set. The value is given in degrees.
- `z [Z degrees]`

This sets the rotation around the z axis of the part considering the center set. The value is given in degrees.

Perspective

```
perspective
{
   zplane: 0;
   focal: 1000;
}
```

Adds focal and plane perspective to the part. Active if `perspective_on` is true. Must be provided if the part is being used by other part as it is perspective target.

- `zplane [unscaled Z value]`
  This sets the z value that is not scaled. Normally this is 0 as that is the z distance.
- `focal [distance]`
  This sets the distance from the focal z plane (zplane) and the camera - i.e. equating to focal length of the camera

Link

```
link
{
   base: "edje,signal" "edje";
   transition: LINEAR 0.2;
   in: 0.5 0.1;
   after: "some_program";
}
```

The link block can be used to create transitions to the enclosing part description state. The result of the above block is identical to creating a program with

```
action: STATE_SET "default"";
signal: "edje,signal";
source: "edje";
```

- `base [signal] [source]`
  Defines the signal and source which triggers the transition to this state. The source parameter is optional here and is filled with the current group's default value if it is not provided.

Program

```
program
{
   // name of the program
   name: "programname";
   // signals which trigger the program
   signal: "signalname";
   // filter incoming signals depending on the sender name
   source: "partname";
   // filter incoming signals depending on the part's state
   filter: "partname" "statename";
   // delay the program by X seconds plus a random time between 0 and Y
   in: 0.3 0.0;
   // action to perform
   action: STATE_SET "statename" state_value;
   // if action is STATE_SET, define a transition from the current to the target state
   transition: LINEAR 0.5;
   // if action is SIGNAL_EMIT, the name of the part which will receive the signal
   target: "partname";
   // run another program after the current one is done
   after: "programname";
   after: "anotherprogram";
}
```

Program

Programs define how your interface reacts to events. Programs can change the state of parts or trigger other events.

- `name [program name]`
  Symbolic name of program as a unique identifier.
- `signal [signal name]`
  Specifies signals that cause the program to run. The signal received must match the specified source to run. There may be several signals, but only one signal keyword per program can be used. Also, there are some predefined signals for touch event handling. The predefined signals are:

- “hold,on”: Holding on the mouse event matching the source that starts the program.
- “hold,off”: Holding off the mouse event matching the source that starts the program.
- “focus,part,in”: Focusing in the matching source that starts the program.
- “focus,part,out”: Focusing out of the matching source that starts the program.
- “mouse,in”: Moving the mouse into the matching source that starts the program.
- “mouse,out”: Moving the mouse out of the matching source that starts the program.

- “mouse,move”: Moving the mouse in the matching source that starts the program.
- “mouse,down,*”: Pressing the mouse button in the matching source that starts the program.
- “mouse,up,*”: Releasing the mouse button in the matching source that starts the program.
- “mouse,clicked,*”: Clicking any mouse button in the matching source that starts the program.
- “mouse,wheel,0,*”: Moving the mouse wheel in the matching source that starts the program. A positive number moves up and a negative number moves down.
- “drag,start”: Starting a drag of the mouse in the matching source that starts the program. This signal works only in the draggable part.
- “drag,stop”: Stopping a drag of the mouse in the matching source that starts the program. This signal works only in the draggable part.
- “drag”: Dragging the mouse in the matching source that starts the program. This signal works only in the draggable part.

- `source [source name]`
  Source of accepted signal. There may be several signals, but only one source keyword per program can be used. For example, source: “button-*”; (signals from any part or program named “button-*” are accepted).

- `filter [part] [state]`
  Filter signals to be only accepted if the part is in state named `[state]`. Only one filter per program can be used. If `[state]` is not given, the source of the event is used instead.

- `in [from] [range]`
  Wait `[from]` seconds before executing the program and add a random number of seconds (from 0 to `[range]`) to the total waiting time.

- `action [type] (param1) (param2) (param3) (param4)`
  Action to be performed by the program. Valid actions (only one can be specified) are:
  - `STATE_SET` : Set “target part” state as “target state”
  - `ACTION_STOP` : Stop the ongoing transition.
  - `SIGNAL_EMIT` : Emit a signal to the application level. The application can register a callback for handling actions based on the EDC state.
  - `DRAG_VAL_SET` : Set a value for the dragable part (x, y values).
  - `DRAG_VAL_STEP` : Set a step for the dragable part (x, y values).
  - `DRAG_VAL_PAGE` : Set a page for the dragable part (x, y values).
  - `FOCUS_SET` : Set the focus to the target group.
  - `FOCUS_SET` : Set the focus to the target group.
  - `PLAY_SAMPLE “sample name” speed (channel)` : Play a music sample clip.
    `PLAY_SAMPLE` 's (optional) channel can be one of:
    - `EFFECT/FX`
    - `BACKGROUND/BG`
    - `MUSIC/MUS`
    - `FOREGROUND/FG`
    - `INTERFACE/UI`
    - `INPUT`
    - `ALERT`
  - `PLAY_TONE “tone name” duration_in_seconds (Range 0.1 to 10.0)` : Play a predefined tone of

a specific duration.
    - `PLAY_VIBRATION "sample name" repeat (repeat count)`
- `transition [type] [length] (interp val 1) (interp val 2) (option)`
  Defines how transitions occur using `STATE_SET` action. `[type]` is the style of the transition and `[length]` is a double specifying the number of seconds in which to preform the transition. Valid types are:
    - `LIN` or `LINEAR`
    - `SIN` or `SINUSOIDAL`
    - `ACCEL` or `ACCELERATE`
    - `DECEL` or `DECELERATE`
    - `ACCEL_FAC` or `ACCELERATE_FACTOR`
    - `DECEL_FAC` or `DECELERATE_FACTOR`
    - `SIN_FAC` or `SINUSOIDAL_FACTOR`
    - `DIVIS` or `DIVISOR_INTERP`
    - `BOUNCE`
    - `SPRING`

`ACCEL_FAC`, `DECEL_FAC` and `SIN_FAC` need the extra optional "interp val 1" to determine the "factor" of curviness. 1.0 is the same as their non-factor counterparts and 0.0 is equal to linear. Numbers higher than 1.0 make the curve angles steeper with a more pronounced curve point.

`DIVIS`, `BOUNCE` and `SPRING` also require "interp val 2" in addition to "interp val 1".

`DIVIS` uses `[val 1]` as the initial gradient start (0.0 is horizontal, 1.0 is diagonal (linear), 2.0 is twice the gradient of linear, etc.). `[val 2]` is interpreted as an integer factor defining how much the value swings outside the gradient before going back to the final resting spot at the end. 0.0 for `[val 2]` is equivalent to linear interpolation. Note that `DIVIS` can exceed 1.0.

`BOUNCE` uses `[val 2]` as the number of bounces (so it is rounded down to the nearest integer value), with `[val 1]` determining how much the bounce decays; 0.0 gives linear decay per bounce and higher values give much more decay.

`SPRING` is similar to bounce; `[val 2]` specifies the number of spring swings and `[val 1]` specifies the decay, but it can exceed 1.0 on the outer swings.

Valid options are:

- `CURRENT` causes the object to move from its current position. Can be used as the last parameter of any transition type.
- `target [target]`
  Program or part on which the specified action acts.
- `after [after]`
  Specifies a program that is run after the current program completes. The source and signal parameters of a program run as an `after` are ignored. Multiple `after` statements can be specified per program.