Enlightenment Foundation Libraries
http://www.enlightenment.org

Architecture & Usage

Sanjeev BA
EFL Enthusiast, Senior Engineer
AS2902.B@SAMSUNG.COM
What is EFL?

• A collection of libraries
• Built by the same team working on Enlightenment
• Built for the purpose of making E17 (Enlightenment 0.17)
• Always focused on staying lean and still providing fanciness
• Almost all development focus sunk into EFL vs E17
• Cover a wide range of functionality due to broad needs
• 26% of code for E17 is E, rest is EFL.
• E17+EFL make up only 50% of code in SVN though
Historical Details

- 1996 – Enlightenment development started
- 1997 – Imaging layer split off into Imlib and Fnlib
- 1997 – Imlib adds GTK+/GDK support
- 1999 – Imlib2 combines images, fonts, alpha channels etc.
- 2001 – Evas (using Imlib2 and OpenGL) first appears

And then EFL really began as more libs were added:
- Ecore, Ebits (later replaced by Edje), Edb (deprecated in favor of Eet), Eina, Embryo, Efreet, EDbus, Ethumb, Emotion, Elementary, Epdf, Eeze.
What's inside

- Canvas scene-graph (Evas)
- OpenGL, OpenGL-ES2.0, Software renderer and more
- Core mainloop, connection, input and glue libraries (Ecore)
- Data codec and storage (Eet)
- Bytecode VM (Embryo)
- Pre-made data objects with scripting, animation etc. (Edje)
- Freedesktop.org standards support (Efreet)
What's inside

- Data structure, modules and base (Eina)
- Dbus integration and wrapping (Edbus)
- Asynchronous I/O (Eio)
- Video playback glue (Emotion)
- Udev hardware detection (Eeze)
- Thumbnailer & cacher (Ethumb)
- Widgets & convenience (Elementary)
So why does this matter?

- EFL is the core toolkit being used in Tizen
- EFL is built for performance and low footprint
  - Still heavy focus on customization and power
- Native apps can use EFL as opposed to shipping their own toolkits
- Smaller footprint for shipping devices
- Continued support
- It's an open source project
- API's all in C, thus easily usable from both C and C++
  - Support for language bindings coming soon.
Where does it lurk?

- HTML5 Apps
  - Browser And HTML5 Runtime
    - Enlightenment (Window Manager & Compositor)
  - Native Applications and Services

- EFL
  - EFL

- libc etc.
- libpng/jpeg etc.
- X11, OpenGL, etc.
- D-Bus, Services

KERNEL
Core OS (Kernel, libc, other system libraries, OpenGL, D-Bus, X11, services etc.)

(CURRENTLY NOT IN TIZEN)
Why EFL?

• Why is EFL being used as opposed to GTK+ or Qt or something else?

• Speed
  • Samsung used GTK+, X11 and DirectFB (in combinations) and once EFL was tried, it soundly beat these hands-down in performance
  • Very fast software rendering (for all occasions)
  • Solid Accelerated OpenGL and OpenGL-ES2.0 support for many years
  • 60fps+ on common smartphones equaling android with higher quality
Why EFL?

• Why is EFL being used as opposed to
  • GTK+ or Qt or something else?
• Benchmarking based on failsafe X11 session (2011)
  • Unity – 168Mb
  • Enlightenment 0.17 – 65Mb
• Roughly similar features and setup
  • Compositor (OpenGL),
  • fullscreen wallpaper,
  • launcher, icons, filemanager, etc.
How is this relevant?

- Mobile devices ship with limited memory
- These devices almost never use swap
- Flash has limited writes, so swap can hurt device lifespan
- Lower end devices may not have GPU's
- Require decent software rendering to make up for it

Samsung Z1
768 MB RAM
480 x 800 pixels (~233 ppi pixel density)
ECORE
Core concepts

• Event driven mainloop
• Rendering (UI)
• Application state management
• Small miscellaneous tasks (non-blocking)
• Support for threaded work
• Added thread models with mainloop begin/end blocks and mainloop call dispatch (from threads).
• More on threading
  • http://docs.enlightenment.org/auto/elementary/threading.html
The Mainloop (Ecore)

1. **Spin in idlers**
2. **Wake up**
3. **Process timeouts**
4. **Process events**
5. **Process jobs**
6. **Go to sleep**
7. **Callback**
   - Idle enterer
   - Timer/Animator
   - Event handler
   - Job
   - Idle exiter

Evlas renders updates
To keep a smooth UI

• Put I/O work or heavy computation into threads
• Use the constructs provided to make this easy
• Keep state in Mainloop consistent
• Only deliver changes as a whole (UI tracks state)
  • automatic within mainloop
• Use Animators, not Timers for animation
• Remember that mainloop is for keeping application state
• Blocking it blocks state (and UI) updates
Threading the Mainloop (Ecore Thread)

Mainloop adds thread job

Queue

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Thread Job

Results returned to Mainloop (result functions run inside Mainloop)
#include <Elementary.h>
static void on_win_del(void *data, Evas_Object *obj, void *event_info) {
    elm_exit();
}
static void on_ok(void *data, Evas_Object *obj, void *event_info) {
    elm_exit();
}
int elm_main(int argc, char **argv) {
    Evas_Object *win, *box, *label, *button;
    win = elm_win_util_standard_add("main", "Hello");
evas_object_smart_callback_add(win, "delete,request", on_win_del, NULL);
    box = elm_box_add(win);
    label = elm_label_add(win);
    elm_object_text_set(label, "Hello out there world");
    elm_box_pack_end(box, label);
    evas_object_show(label);
    button = elm_button_add(win);
    elm_object_text_set(button, "OK");
    elm_box_pack_end(box, button);
    evas_object_show(button);
evas_object_smart_callback_add(button, "clicked", on_ok, NULL);
    elm_win_resize_object_add(win, box);
evas_object_show(box);
evas_object_show(win);
    elm_run();
}
ELM_MAIN();
Hello EFL
(in C)

#include <Elementary.h>

static void on_win_del(void *data, Evas_Object *obj, void *event_info) {
  elm_exit();
}

static void on_ok(void *data, Evas_Object *obj, void *event_info) {
  elm_exit();
}

int elm_main(int argc, char **argv) {
  Evas_Object *win, *box, *label, *button;
  win = elm_win_util_standard_add("main", "Hello");
  evas_object_smart_callback_add(win, "delete,request", on_win_del, NULL);
  box = elm_box_add(win);
  label = elm_label_add(win);
  elm_object_text_set(label, "Hello out there world");
  elm_box_pack_end(box, label);
  evas_object_show(label);
  button = elm_button_add(win);
  elm_object_text_set(button, "OK");
  elm_box_pack_end(box, button);
  evas_object_show(button);
  evas_object_smart_callback_add(button, "clicked", on_ok, NULL);
  elm_win_resize_object_add(win, box);
  evas_object_show(box);
  evas_object_show(win);
  elm_run();
}

ELM_MAIN();
Hello EFL
(in C)

```c
#include <Elementary.h>
static void on_win_del(void *data, Evas_Object *obj, void *event_info) {
    elm_exit();
}
static void on_ok(void *data, Evas_Object *obj, void *event_info) {
    elm_exit();
}
int elm_main(int argc, char **argv) {
    Evas_Object *win, *box, *label, *button;
    win = elm_win_util_standard_add("main", "Hello");
    evas_object_smart_callback_add(win, "delete,request", on_win_del, NULL);
    box = elm_box_add(win);
    label = elm_label_add(win);
    elm_object_text_set(label, "Hello out there world");
    elm_box_pack_end(box, label);
    evas_object_show(label);
    button = elm_button_add(win);
    elm_object_text_set(button, "OK");
    elm_box_pack_end(box, button);
    evas_object_show(button);
    evas_object_smart_callback_add(button, "clicked", on_ok, NULL);
    elm_win_resize_object_add(win, box);
    evas_object_show(box);
    evas_object_show(win);
    elm_run();
}
ELM_MAIN();
```
#include <Elementary.h>

static void on_win_del(void *data, Evas_Object *obj, void *event_info) {
    elm_exit();
}

static void on_ok(void *data, Evas_Object *obj, void *event_info) {
    elm_exit();
}

int elm_main(int argc, char **argv) {
    Evas_Object *win, *box, *label, *button;
    win = elm_win_util_standard_add("main", "Hello");
    evas_object_smart_callback_add(win, "delete,request", on_win_del, NULL);
    box = elm_box_add(win);
    label = elm_label_add(win);
    elm_object_text_set(label, "Hello out there world");
    elm_box_pack_end(box, label);
    evas_object_show(label);
    button = elm_button_add(win);
    elm_object_text_set(button, "OK");
    elm_box_pack_end(box, button);
    evas_object_show(button);
    evas_object_smart_callback_add(button, "clicked", on_ok, NULL);
    elm_win_resize_object_add(win, box);
    evas_object_show(box);
    evas_object_show(win);
    elm_run();
}

ELM_MAIN();
Hello EFL
(in C)

- $ gcc hello.c -o hello `pkg-config --cflags --libs elementary`
- $ ./hello
EVAS
What is a scene graph? (Evas)

- Tracks state of all display objects
  - Position, size, visibility, color, properties etc.
- Handles rendering of each object
  - Loading fonts, images, rendering glyphs, scaling, fading etc.
- Handles minimizing of rendering
  - Only update areas changed
- If changes obscured, reduce to a NOP
  - Optimize rendering
- Abstract to OpenGL, software, or anything else
What is a scene graph? (Evas)

- Allows you to build your own composite objects
  - Creates parent/child relationship
- Is used throughout EFL to build widgets etc.
- Handles input direction and event callbacks
- Text formatting & layout
Putting together objects
Abstracting rendering

Canvas core

Hello out there

Input device events
Image data, fonts etc.

Canvas state changes
Rendering command

General Rendering

Select rendering engine at runtime

Software rendering API
OpenGL Rendering API

Software core
OpenGL core

OUTPUT (screen/window)
Automated update handling

Start here

Hello out there
Automated update handling

Next frame is here

Hello out there
Automated update handling

Calculate actual update region deltas (up to each engine to implement)
Automated update handling

Only draw updated regions (up to each engine to...
Automated update handling

Result

Hello out there
Multiple output paths

- X11 (OpenGL, Xlib & XCB)
- Wayland (OpenGL & SHM)
- Raw Framebuffer
- Memory buffers
- PS3 Native
- SDL (OpenGL)
- Windows (32/64/CE) (GDI & DirectDraw)
- ... others too
EDJE
Pre-made objects for designers (Edje)

- Edje allows a designer to store objects in files
- Pre-made layout with rules and reactions to events
- Stored separately to code in binary files for runtime replacement
- Fast & compact random access designed for realtime use
- All layout, image data, etc. etc. all in 1 file (zero-unpacking)
- Intended for designers & developers to work independently
- Supports scalable and resizeable layouts
- Provides the core ability to re-theme and entire UI or OS
How it works

Application
- Signal emits
- Messages
- Signal callbacks
- Messages
- Queries
- Create & control Objects
- Get event callbacks

Mainloop
- Controls
- Swallows
- Text etc.
- Layout rules, parameters & states
- Images & fonts
- Event reaction rules

Edje File

Edje
- Signals
- Messages
- Queries
- Controls
- Text etc.
An example

collections {
  group { name: "hello";
    images {
      image: "plant.jpg" LOSSY 80;
      image: "shadow.png" COMP;
    }
    parts {
      part { name: "bg";
        description { state: "default" 0.0;
          aspect: 1.0 1.0; aspect_preference: NONE;
          image.normal: "plant.jpg";
        }
      }
      part { name: "label"; type: TEXT; scale: true;
        description { state: "default" 0.0;
          text { 
            font: "Sans"; size: 20;
            text: "Hello World!";
          }
        }
      }
      part { name: "shadow";
        description { state: "default" 0.0;
          image.normal: "shadow.png";
        }
      }
    }
  }
}
ELEMENTARY
So what is Elementary?

- A widget set built on top of the lower-level EFL layers
- Brings coherent policy and consistency to widgets
- Pre-made common widgets most applications need
- Central theme setup so applications look consistent
- Utilities saving extra footwork by the developer
- Touch friendly design
- Scaling of UI from the get-go
- Also adjusts for input resolution (finger vs mouse etc.)
So what is Elementary?

- It can be seamlessly merged with lower level objects
- Programmer can use Elementary containers or hand-arrange widgets and control them
- Since all objects can be stacked and layered, so can elementary widgets
- Widgets can be transformed like any object
- Handles dealing with IME (Virtual keyboard) for you
- Does many other useful things to save you time
Results with Elementary
• EMOTION
Video & Sound in your world

- Gives you a high level API to include video
- Abstracts to different video decode back-ends
- Optimizes decode via YUV paths or video overlay
- Simple to use
Simple Video

- `Evas_Object *vid = emotion_object_add(canvas);`
- `emotion_object_init(vid, NULL);`
- `emotion_object_file_set(vid, "file.avi");`
- `evas_object_resize(vid, 640, 360);`
- `emotion_object_play_set(vid, EINA_TRUE);`
- `evas_object_show(vid);`
How it works

- Application
- Evas
- Mainloop
- Emotion Core
- Media Codec (Gstreamer/Xine/Generic)
  - Thread
  - Thread
  - Thread
- Media file or stream
EET
Garbage in, garbage out

struct Data {
    const char *name;
    ...
}

010001110101011010101
10101

Binary Data (File or buffer)

Eet (Library)

Application (RAM)

Text file Stdout

group “Data” struct {
    value “name” string: “Bob”;}

Eet (Cmdline)
XML/JSON ... for C programmers

• Parsing text is painful
• Parsing correctly without bugs, overflows is harder
• Most programmers hate parsing
• XML, JSON etc. optimized for editing, not runtime
• Programs read/write data 1000x more than humans
• So optimize for the common use case, not the uncommon one
• Make it as easy 1-liners for C code to load or save data
• Edje, Enlightenment config, Elementary config built on EET
Flexible, portable and robust

- Allows you to store data in a file (key/value pair)
- Random access read optimized
- Data can be any binary, image, string or struct encoded
- Compresses separate keys (like zip)

- Allows you to en/decode structs to buffers (for network)
- Provides a protocol buffer handler for decodes
- Files and data all platform agnostic (portable)
- Structs encoded with nested key & types for robustness
•EDBUS EFREET EINA ETHUMB EEZE EMBRYO EIO ...
And the saga continues

- More EFL libraries with no time to mention them
- Expanding libs and scope on a daily basis
QnA

• Enlightenment Foundation Libraries
  • http://www.enlightenment.org
    • Join our
    • IRC : #edev, #e
  • Mailing Lists
  • https://lists.sourceforge.net/lists/listinfo/enlightenment-devel