SOSCON
삼성 오픈소스 컨퍼런스
SAMSUNG OPEN SOURCE CONFERENCE
OPEN YOUR UNIVERSE WITH SOSCON

Eo : EFL object system

Samsung Electronics
Jee-Yong Um

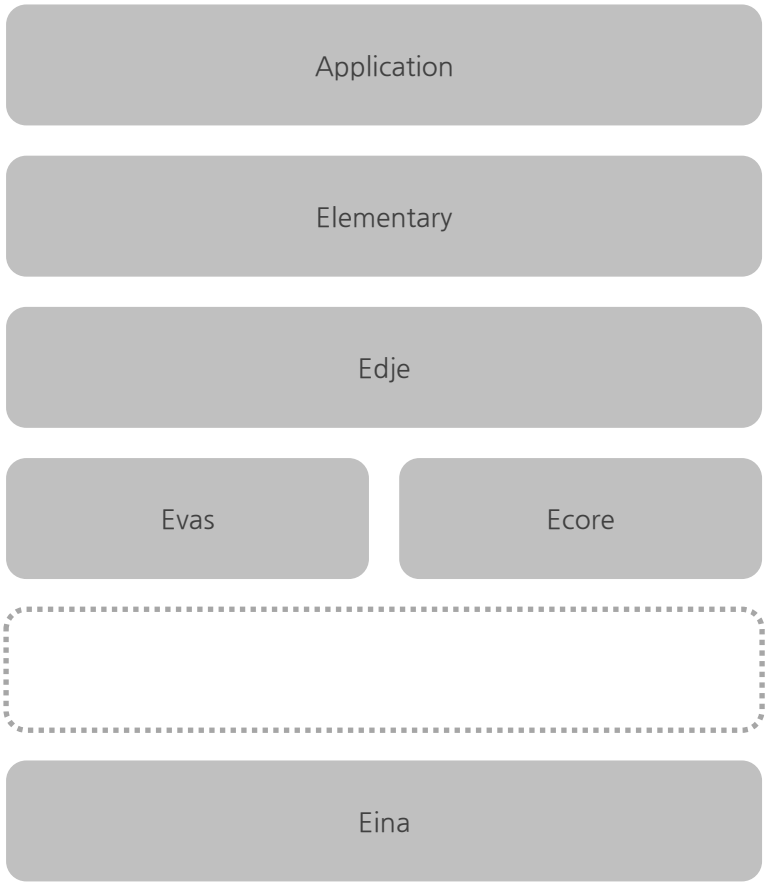
28 Oct 2015

Complex ‘E’ world

- In ‘E’ world, there have been several modules like Evas, Ecore or Elementary.
- Each module has its own object system.

Evas	Ecore	Elementary
Evas_Object	Ecore_Animator	Elm_Widget
Evas_Rectangle	Ecore_Timer	Elm_Layout
Evas_Image	Ecore_Idler	Elm_Button
Evas_Textblock	Ecore_Job	Elm_Hoversel
…	…	…

- User’s reponsibility to free the object
- Difficult to automate the language binding
- Lack of new language features like interface, mixin, class extension etc.

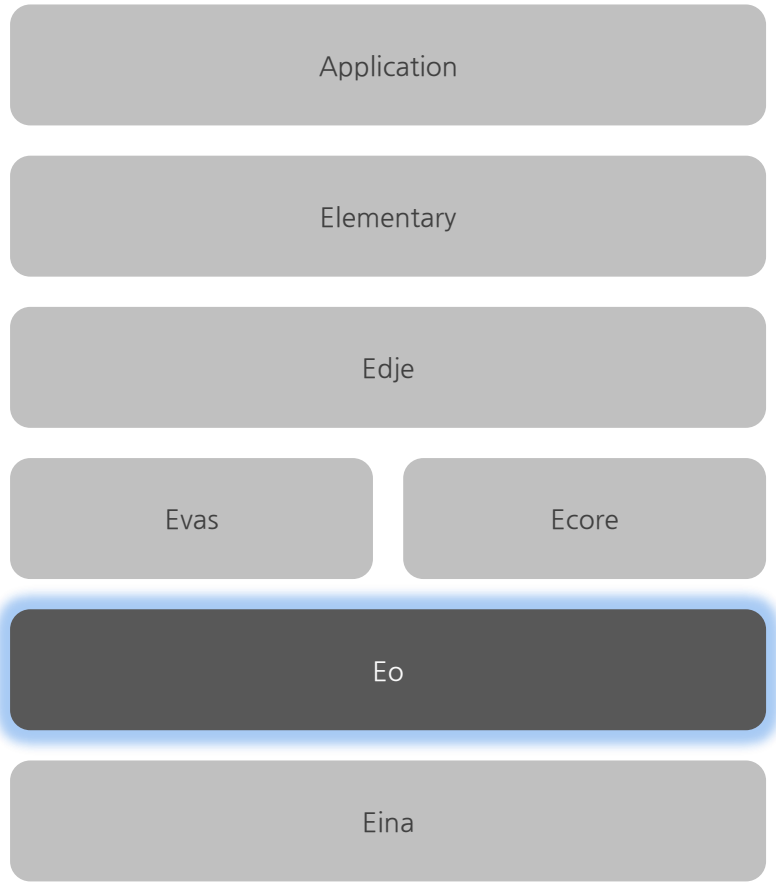


Another C object system for EFL

- Eo provides objects in plain-old-C with next features.

- Inheritance
- Interfaces
- Mixins
- Multiple inheritance
- Method / property overriding
- Properties
- All object internals are opaque
- Reference counting
- Callbacks (all objects)
- Cross references
- Parent / child object hierarchy
- Weak references
- Key / value attachment on all objects
- Code generation / maintenance to avoid boilerplate
    - Define your classes in Eo files cleanly
- Multiple language support (beyond C) for bindings
    - Done by code generation and base type support
    - Currently C++ and Lua
        - C++ bindings use C ABI, not C++ ABI (fewer problems)
    - Plans to add Python and JS (v8)
- Runtime type safety for objects
- Runtime method / property call safety (if not supported becomes NOOP)
- Object indirection to remove pointer bugs
- Multi-call per object de-reference (lowers object access overhead)
- Construction finalizers allowing calls during construction for lower setup overheads

ref) https://phab.enlightenment.org/phame/live/1/post/yet_another_c_object_model_but_better/



To put it simply, Eo means EFL Object

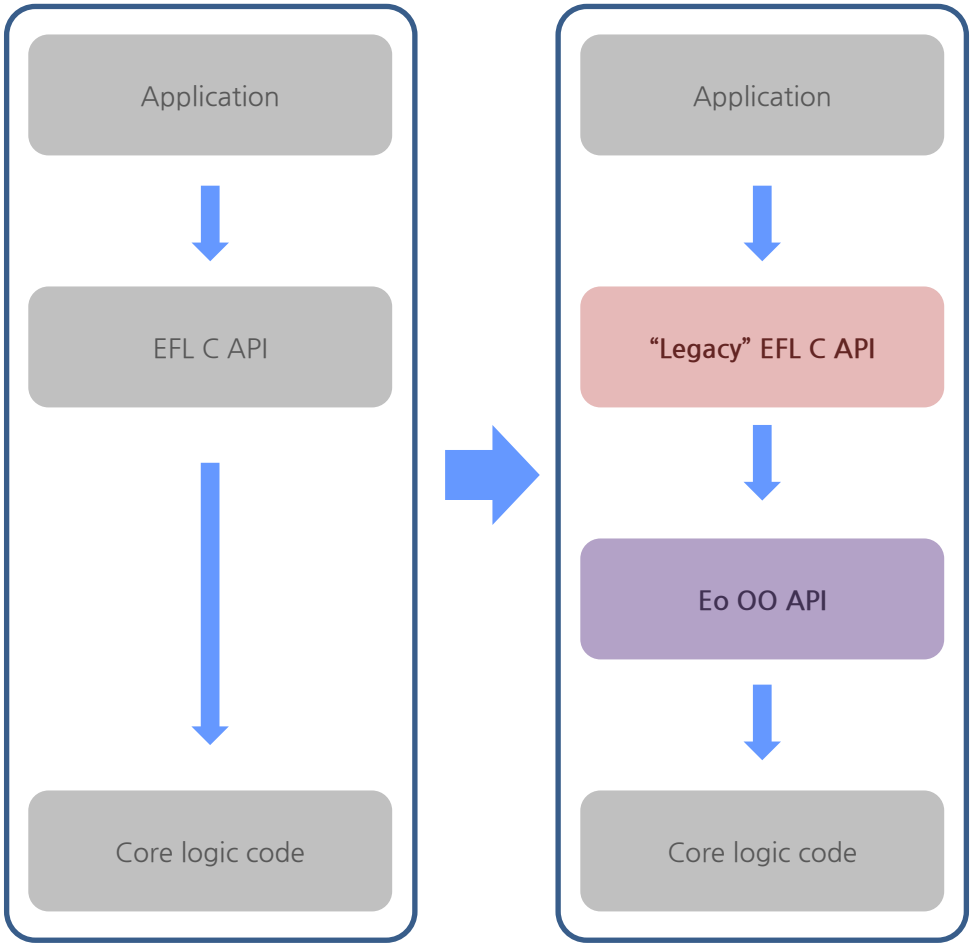
- Added OO layer below old C API
  - Keep ABI/API compatibility
  - Add cleaner & consistent OO API
- Added safety layer for object access
  - Crashes in magic check go away
- Before

  elm_layout_text_set(obj, part, text);
- After

  elm_layout_text_set(obj, part, text); (Legacy)

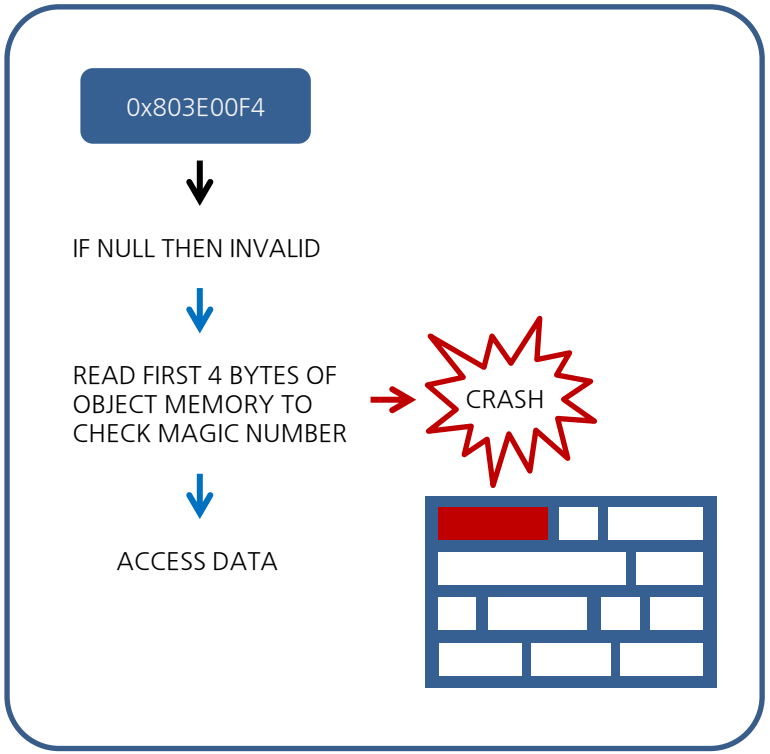
  → eo_do(obj, ret = elm_obj_layout_text_set(part, text)); (Eo)

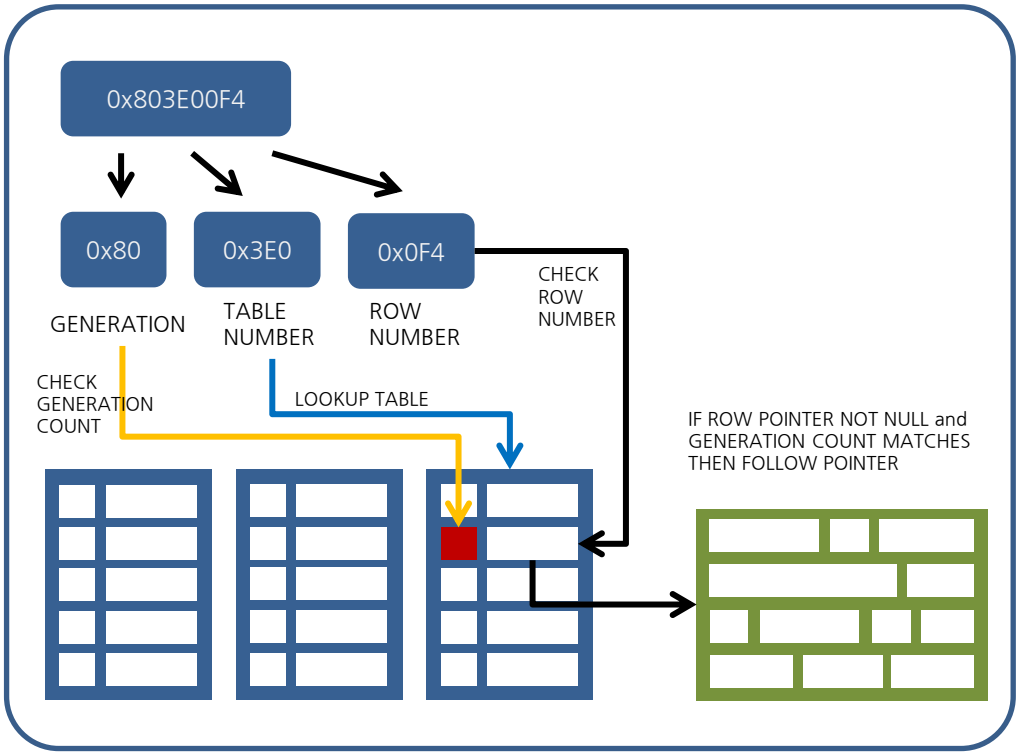
  →_elm_layout_text_set(obj, pd, part,text);



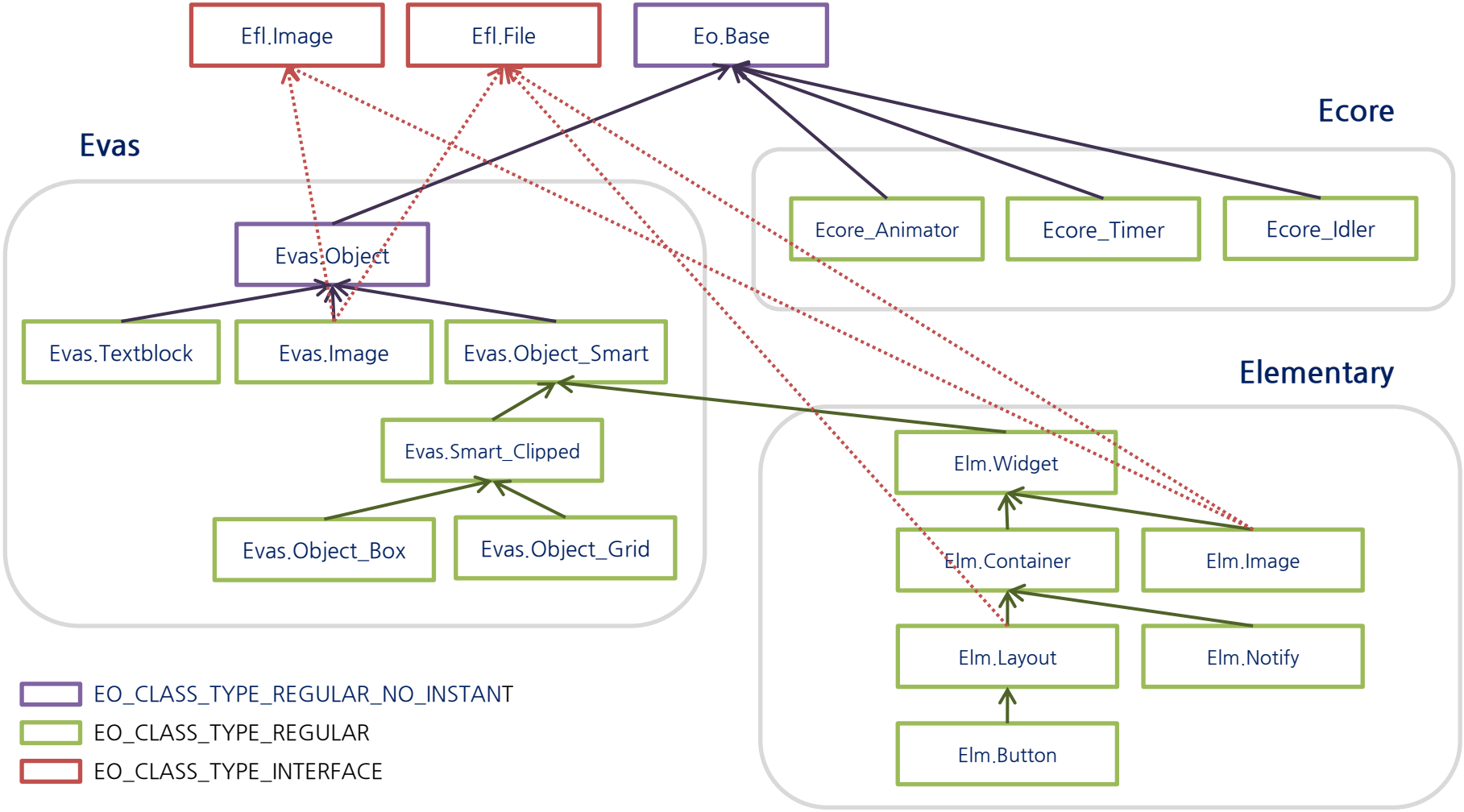
No more pointers

- In C and C++ most objects are pointers (Qt, GTK+, EFL)
- However, Eo * does not mean specific address in memory any more.



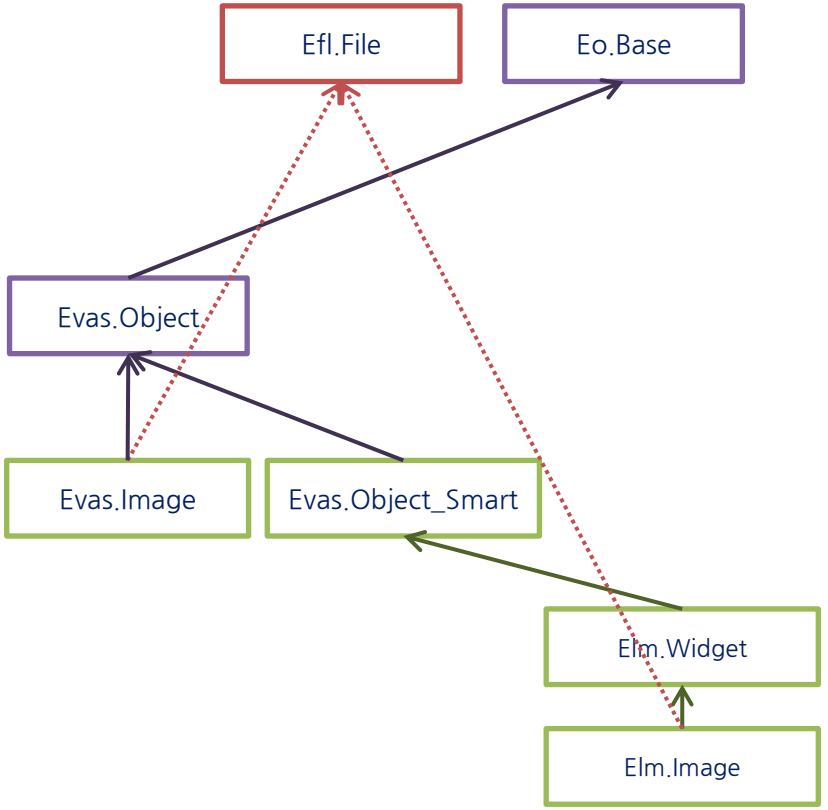


All objects in EFL modules are Eo objects



Example 1

- Legacy
  - Evas_Object *obj1 = evas_object_image_add(e);
  - evas_object_image_file_set(obj1, “image.png”, NULL);
  - Evas_Object *obj2 = elm_image_add(parent);
  - elm_image_file_set(obj2, “image.png”, NULL);
- Eo
  - interfaces Efl.File ()
    - efl_file_set(const char *file, const char *key);
  - class Evas.Image (Evas.Object, Efl.File, …)
    - _evas_image_efl_file_file_set();
  - class Elm.Image (Elm.Widget, Efl.File, …)
    - _elm_image_efl_file_file_set();
  - eo_do(obj1, efl_file_file_set(“image.png”, NULL));
  - eo_do(obj2, efl_file_file_set(“image.png”, NULL));



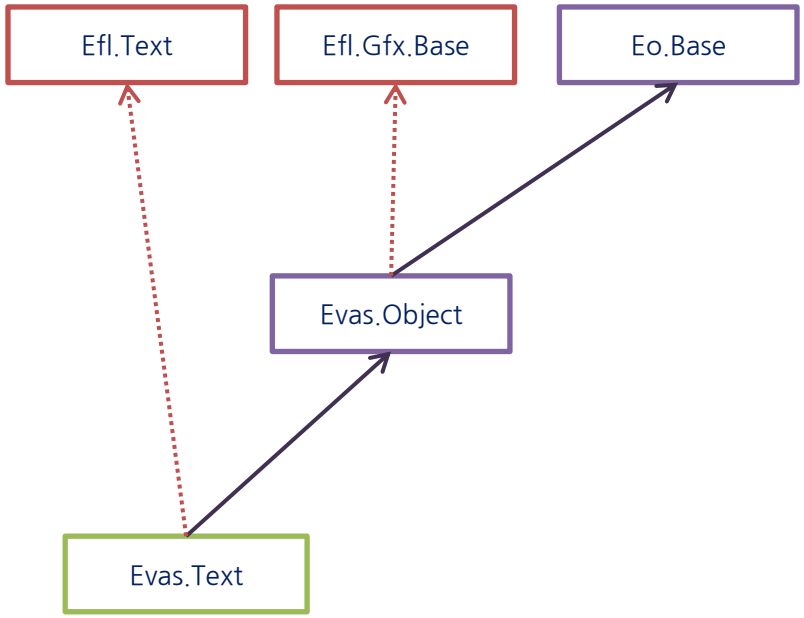
Example 2

- Legacy
  - Evas_Object *obj = evas_object_text_add((Evas *e));
  - evas_object_move(obj, 100, 100);
  - evas_object_resize(obj, 200, 200);
  - evas_object_text_text_set(obj, “text”);

- Eo
  - abstract Evas.Object (Eo.Base, Efl.Gfx.Base, …)
  - interfaces Efl.Text ()
    - efl_text_set(const char *text);
  - class Evas.Text (Evas.Object, Efl.Text, …)

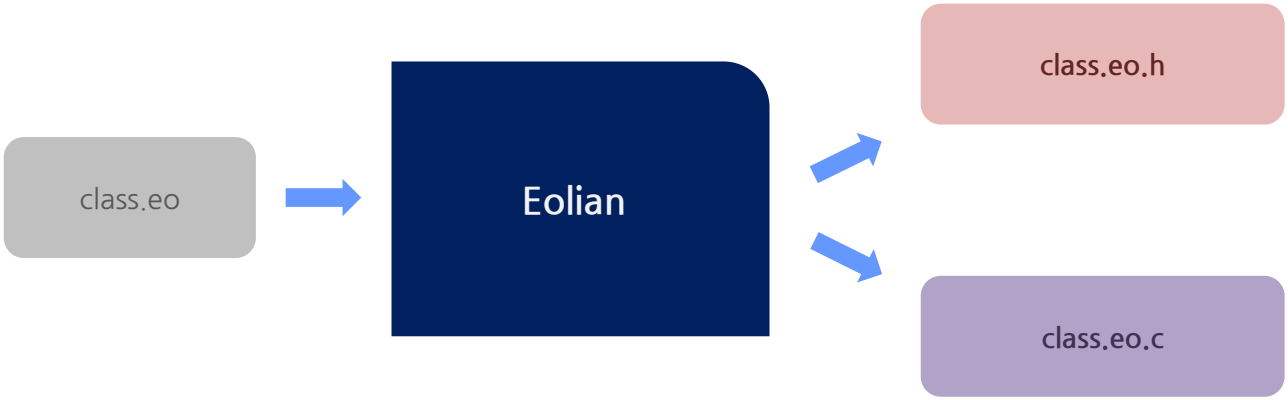
  - Evas_Object *obj = eo_add(EVAS_TEXT_CLASS, e);
  - eo_do(obj, efl_gfx_position_set(100, 100),
    efl_gfx_size_set(200, 200),
    efl_text_set(“text”));

  - _evas_object_efl_gfx_base_position_set();
    _evas_object_efl_gfx_base_size_set();
    _evas_text_efl_text_text_set();



To create Eo class manullay, too HARD!

- Eo APIs are combination of numerous macro functions and common c functions.
- To write eo class code by hand is almost insanely complex work.
- Fortunately, eolian will make code for eo class automatically, user needs to write core logic code only.



- Eo API Declaration
- Macro to get the class

- Class structure
- Fuction Table
- Class constructor
- Legacy API definition
- Implementation function declaration

eo file contains the definition of eo class

- language neutral syntax (allows to ease binding languages, but costs because of learning new syntax)

```
class class_name (class1, class2 , interface1)
{
        methods {
                // define public methods of the class
        }
        implements {
                // implements method from the base class or interface
        }
        events {
                // defines event for the class
        }
}
```

class sample

```
class Elm.Genlist (Elm.Layout, Elm_Interface_Scrollable, ···) {
   data: Elm_Genlist_Data;
   methods {
      @property homogeneous {
         set {
         }
         get {
         }
         values {
            homogeneous: bool;
         }
      }
      item_append {
         return: Elm.Widget_item *;
         params {
            @in itc: const(Elm_Genlist_Item_Class)*;
            ···
         }
      }
   }
   implements {
      Eo.Base.constructor;
      Evas.Object_Smart.add;
      Elm.Layout.sizing_eval;
      ···
   }
   events {
      item,focused;
      ···
   }
}
```

```
typedef struct {

   ···

} Elm_Genlist_Data;
```

```
void
elm_genlist_homogeneous_set(Evas_Object *obj, Eina_Bool homogeneous);

Eina_Bool
elm_genlist_homogeneous_get(Evas_Object *obj);
```

```
Elm_Object_Item *elm_genlist_item_append(obj, itc, ···);
```

```
eo_base_constructor(obj); → _elm_genlist_eo_base_constructor();
evas_object_smart_add(obj) → _elm_genlist_evas_object_smart_add();
elm_layout_sizing_eval(obj) → _elm_genlist_elm_layout_sizing_eval();
```

```
eo_do(obj,
eo_event_callback_call(ELM_GENLIST_EVENT_ITEM_FOCUSED, eo_it));
```

Steps to add Eina_Bool foo(int i, char *str) API in Elm_Button Class

Add the API in the elm_button.eo file in the method section

elm_button.eo

```
foo {
        params {
                @in i: int;
                @out str: char;
        }
        return Eina_Bool;
}
```

The Eolian Generator will create a private implementation function with signature
Eina_Bool _elm_button_foo(Eo *obj, Elm_Button_Data *sd, int i, char *str);

Add the private implementation function in elm_button.c file

elm_button.eo

```
EOLIAN static Eina_Bool
_elm_button_foo(Eo *obj, Elm_Button_Data *sd, int i, char *str)
{
   // Add your code here
}
```

To update the implementation of existing API , find the private implementation function and update the code.

NOTE : All the private implementation function has signature as
ret_value Class_Name_# API_Name(Eo * obj, Class_Name_Data *sd , param_list…);

THANK YOU!