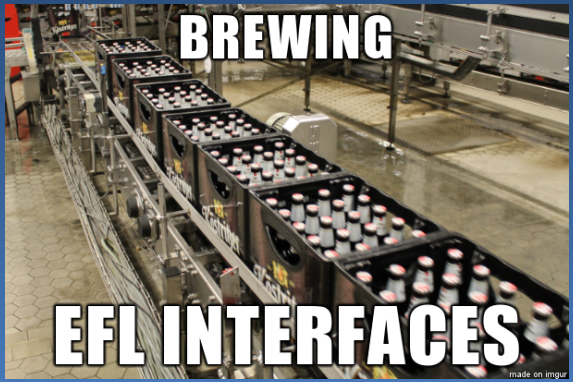
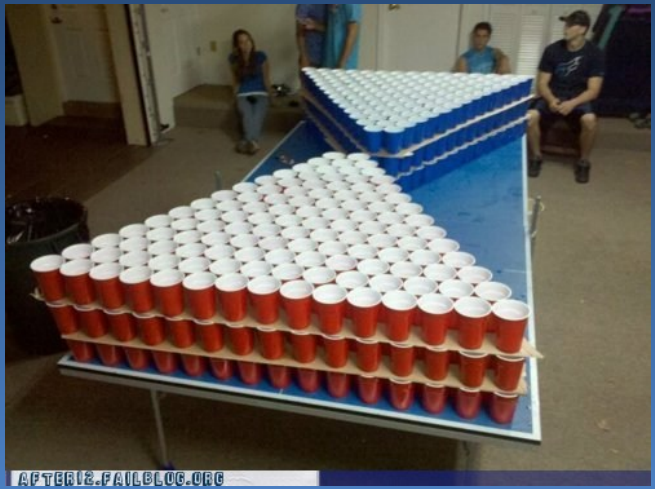
Review of Eolian, Eo, Bindings, Interfaces and What's to Come



Tom Hacohen <tom.hacohen@samsung.com>
Daniel Kolesa <d.kolesa@samsung.com>

What is Eo?



Existing solutions?

Existing solutions?

- GObjecct

Existing solutions?

- GObjecct
- libobjc

Existing solutions?

- GObjecct
- libobjc
- systemd

Why roll our own?

How we did it?

How we did it?

- A lot of prototyping $\rightarrow$ Eo1

How we did it?

- A lot of prototyping $\rightarrow$ Eo1
- A lot of complaints and hate-mail $\rightarrow$ Eo2

How we did it?

- A lot of prototyping $\rightarrow$ Eo1
- A lot of complaints and hate-mail $\rightarrow$ Eo2
- Got annoyed with writing boiler-plate $\rightarrow$ Eolian

How we did it?

- A lot of prototyping → Eo1
- A lot of complaints and hate-mail → Eo2
- Got annoyed with writing boiler-plate → Eolian
- Eolian didn't cover everything we needed → Eolian (current iteration)

Unifying the EFL API (AKA EFL Interfaces)

Unifying the EFL API (AKA EFL Interfaces)

Before:

```
evas_object_image_file_set(obj, "blah.png", "key");
edje_object_file_set(obj, "blah.edj", "group");

evas_object_del(obj);
ecore_timer_del(obj);
ecore_animator_del(obj);
```

Unifying the EFL API (AKA EFL Interfaces)

Before:

```
evas_object_image_file_set(obj, "blah.png", "key");
edje_object_file_set(obj, "blah.edj", "group");

evas_object_del(obj);
ecore_timer_del(obj);
ecore_animator_del(obj);
```

After:

```
eo_do(obj, efl_file_set("blah.file", "key"));

eo_del(obj);
```

Object lifetime

Object lifetime

- `eo_add()` has a C friendly refcount handling

Object lifetime

- `eo_add()` has a C friendly refcount handling
- `eo_add()` $\leftrightarrow$ `eo_del()`

Object lifetime

- `eo_add()` has a C friendly refcount handling
- `eo_add()` $\leftrightarrow$ `eo_del()`
- `eo_ref()` $\leftrightarrow$ `eo_unref()`

Safety features

Safety features

- Pointer indirection (eo id)

Safety features

- Pointer indirection (eo id)
- Object type checks when calling functions

Safety features

- Pointer indirection (eo id)
- Object type checks when calling functions
- Default return values on errors

Safety features

- Pointer indirection (eo id)
- Object type checks when calling functions
- Default return values on errors

Safety features

- Pointer indirection (eo id)
- Object type checks when calling functions
- Default return values on errors

For example:

```
ERR<32099>:eo eo_ptr_indirection.x:287 ↩
    _eo_obj_pointer_get() obj_id 0x13371337 is not ↩
    pointing to a valid object. Maybe it has already been ↩
    freed.
ERR<32124>:eo eo_private.h:283 _eo_unref() Object ↩
    0xDEADBEEF already deleted.
```

Class types

Class types

- Normal class

Class types

- Normal class
- Non instantiate-able class

Class types

- Normal class
- Non instantiate-able class
- Interface

Class types

- Normal class
- Non instantiate-able class
- Interface
- Mixin

Using Eo

Using Eo

- `eo_do(obj, efl_file_set("file.eet", "key"));`

Using Eo

- `eo_do(obj, efl_file_set("file.eet", "key"));`
- `if (eo_do(obj, elm_widget_enabled_get()))`

Using Eo

- `eo_do(obj, efl_file_set("file.eet", "key"));`
- `if (eo_do(obj, elm_widget_enabled_get()))`
- `eo_do(obj, visible = elm_widget_visibility_get(), ↩ elm_widget_visibility_set(!visible));`

Using Eo

- `eo_do(obj, efl_file_set("file.eet", "key"));`
- `if (eo_do(obj, elm_widget_enabled_get()))`
- `eo_do(obj, visible = elm_widget_visibility_get(), ↩ elm_widget_visibility_set(!visible));`
- `eo_do(obj, elm_widget_visibility_set(!elm_widget_visibility_get()));`

Using Eo

- `eo_do(obj, efl_file_set("file.eet", "key"));`
- `if (eo_do(obj, elm_widget_enabled_get()))`
- `eo_do(obj, visible = elm_widget_visibility_get(), ↩`
  `elm_widget_visibility_set(!visible));`
- `eo_do(obj, elm_widget_visibility_set(!elm_widget_visibility_get()));`
- ```
  static void _size_multiply(double f)
  {
      int w, h;
      evas_object_geometry_get(NULL, NULL, &w, &h);
      evas_object_geometry_set(NULL, NULL, w * f, h * f);
  }
  eo_do(obj, _size_multiply(3.5));
  ```
  ```

The Eolian library

But writing objects in C is tedious!

But writing objects in C is tedious!

- The answer: Eolian

But writing objects in C is tedious!

- The answer: Eolian
- Eolian parses Eo API declarations

But writing objects in C is tedious!

- The answer: Eolian
- Eolian parses Eo API declarations
- Eolian allows for automated binding generators

But writing objects in C is tedious!

- The answer: Eolian
- Eolian parses Eo API declarations
- Eolian allows for automated binding generators
- Eolian is meant to be familar for everyone

A new format?

A new format?

- Language independent $\rightarrow$ easy bindings

A new format?

- Language independent $\rightarrow$ easy bindings
- Familiar syntax $\rightarrow$ easy to pick up

A new format?

- Language independent $\rightarrow$ easy bindings
- Familiar syntax $\rightarrow$ easy to pick up
- Easy to read and write

A new format?

- Language independent $\rightarrow$ easy bindings
- Familiar syntax $\rightarrow$ easy to pick up
- Easy to read and write
- Declarative and descriptive

```
class Namespace.Class (inherits) {
    methods { ... }
    properties { ... }
    events { ... }
    implements { ... }
    constructors { ... }
}

type Type_Name: Type_Def;
struct Struct_Name { ... }
enum Enum_Name { ... }
```

```
methods {
    method_name @class @protected {
        params {
            @in int x;
            @out const(char) *y;
        }
        return: own(char*);
    }
}
```

```
properties {
    property_name {
        keys {
            list<int> *x;
        }
        values {
            int v;
        }
        get {}
        set {}
    }
}
```

Generators!

Generators!

- Initial generator: C

Generators!

- Initial generator: C
- Further generators in core EFL: C++ and Lua

Generators!

- Initial generator: C
- Further generators in core EFL: C++ and Lua
- Third party generators: Python, efforts being put into Rust, OCaml

Generators!

- Initial generator: C
- Further generators in core EFL: C++ and Lua
- Third party generators: Python, efforts being put into Rust, OCaml
- Future generators include JavaScript and others

The Eolian library

The Eolian library

- C API: simple and easy to use

The Eolian library

- C API: simple and easy to use
- Minimum of non-standard data types $\rightarrow$ easy to bind

The Eolian library

- C API: simple and easy to use
- Minimum of non-standard data types $\rightarrow$ easy to bind
- Not only for generators (IDEs...)

The Eolian library

- C API: simple and easy to use
- Minimum of non-standard data types $\rightarrow$ easy to bind
- Not only for generators (IDEs...)
- Simple database

However…

However...

- Some things still missing

However...

- Some things still missing
- Documentation?

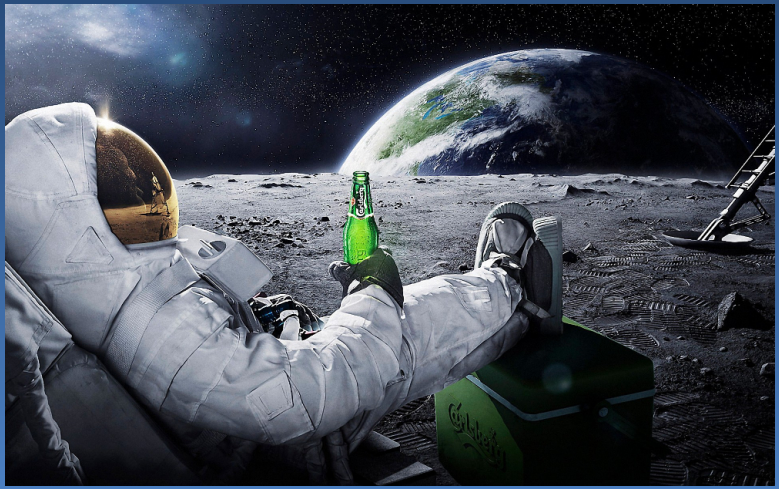
However...

- Some things still missing
- Documentation?
- Value ownership

However...

- Some things still missing
- Documentation?
- Value ownership
- And possibly others

Lua review



The Lua generator

The Lua generator

- Third generator (after C and C++) $\rightarrow$ Lua

The Lua generator

- Third generator (after C and C++) $\rightarrow$ Lua
- Built around our Elua application runtime

The Lua generator

- Third generator (after C and C++) $\rightarrow$ Lua
- Built around our Elua application runtime
- Itself a Lua application

The Lua generator

- Third generator (after C and C++) $\rightarrow$ Lua
- Built around our Elua application runtime
- Itself a Lua application
- Helped the Eolian C library go forward

The FFI

The FFI

- LuaJIT C FFI $\rightarrow$ simple bindings

The FFI

- LuaJIT C FFI $\rightarrow$ simple bindings
- Simple bindings $\rightarrow$ easy debugging

The FFI

- LuaJIT C FFI $\rightarrow$ simple bindings
- Simple bindings $\rightarrow$ easy debugging
- Also, no compiled modules

The FFI

- LuaJIT C FFI $\rightarrow$ simple bindings
- Simple bindings $\rightarrow$ easy debugging
- Also, no compiled modules
- Also, simple generation

The infrastructure

The infrastructure

- Handwritten Eo bindings

The infrastructure

- Handwritten Eo bindings
- No object wrappers, FFI metatypes instead

The infrastructure

- Handwritten Eo bindings
- No object wrappers, FFI metatypes instead
- Builtin method dispatch via metatables

The infrastructure

- Handwritten Eo bindings
- No object wrappers, FFI metatypes instead
- Builtin method dispatch via metatables
- Eo inheritance and reference management

The infrastructure

- Handwritten Eo bindings
- No object wrappers, FFI metatypes instead
- Builtin method dispatch via metatables
- Eo inheritance and reference management
- No wrappers $\rightarrow$ fast, simple, no tracking

```
local util = require("util")
... more utils follow ...

local M, __lib = ...

local __class, __body

-- init func registers the class with __body
cutil.init_module(init_func, shutdown_func)

ffi.cdef [[ C API definitions in C syntax ]]

__body = { ... wrapper funcs over C API funcs ... }

M.My_Class = function(parent, ...)
    ... construct the instance and return it, like C ↩
        would ...
end
```

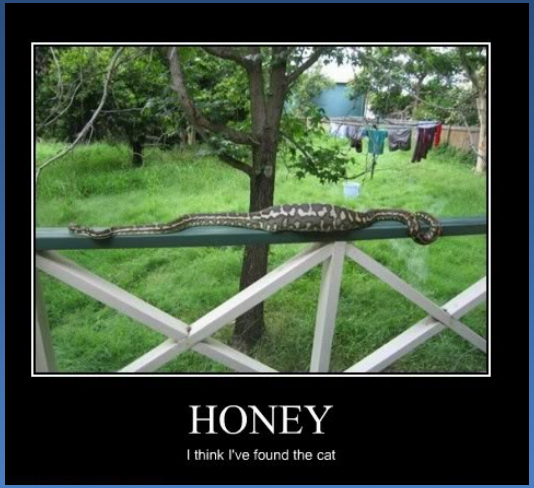
```
local elm = require("elm")

local win = elm.Window(nil, "mywin", elm.win_type.BASIC)
win.autodel = false
win.size = { 500, 500 }

win:connect("delete,request", function() ... event ... ↩
    end)
win:resize_object_add(obj)

...
```

Python



The Python generator

The Python generator

- Handwritten Eo bindings

The Python generator

- Handwritten Eo bindings
- A Python script generates Cython code

The Python generator

- Handwritten Eo bindings
- A Python script generates Cython code
  - So compiled. . . :(

The Python generator

- Handwritten Eo bindings
- A Python script generates Cython code
  - So compiled. . . :(
  - Will be fixed. . .

The Python generator

- Handwritten Eo bindings
- A Python script generates Cython code
  - So compiled. . . :(
  - Will be fixed. . .

The Python generator

- Handwritten Eo bindings
- A Python script generates Cython code
  - So compiled. . . :(
  - Will be fixed. . . FFI!

The Python bindings

The Python bindings

- Native Python classes and inheritance

The Python bindings

- Native Python classes and inheritance
- Native Python properties

The Python bindings

- Native Python classes and inheritance
- Native Python properties
- Native Python modules

The Python bindings

- Native Python classes and inheritance
- Native Python properties
- Native Python modules
- Everything feels native

How to use?

How to use?

- Native Python...

How to use?

- Native Python...
- Properties

```
from elementary import Win
win = Win(parent, "win name", Win.ELM_WIN_BASIC)
win.size = (600, 600)
win.visibility = True
```

How to use?

- Native Python...
- Properties

```
from elementary import Win
win = Win(parent, "win name", Win.ELM_WIN_BASIC)
win.size = (600, 600)
win.visibility = True
```

- Methods

```
win.resize_object_add(obj)
```

How to use?

- Native Python...
- Properties

```
from elementary import Win
win = Win(parent, "win name", Win.ELM_WIN_BASIC)
win.size = (600, 600)
win.visibility = True
```

- Methods

```
win.resize_object_add(obj)
```

- Callbacks

```
obj.connect("mouse,down", some_callable)
```

But what about the current bindings?

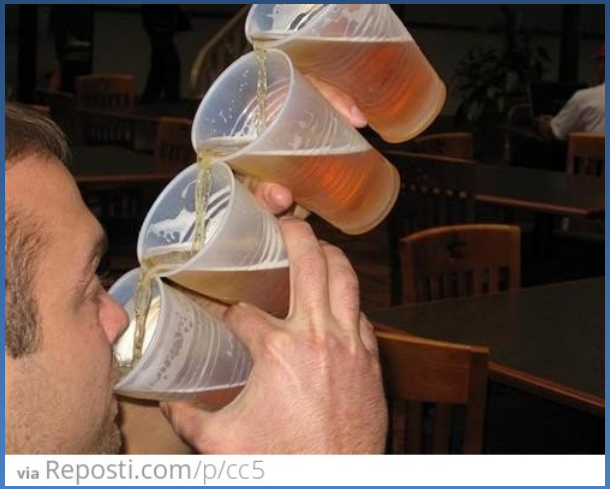
But what about the current bindings?

- Incompatible. :(

But what about the current bindings?

- Incompatible. :(
- Kai wants to write a compatibility layer

DEMOS



What's next?

More bindings!

Making bindings embeddable

Use Eolian to do more

Use Eolian to do more

- The EFL GUI builder - already there

Use Eolian to do more

- The EFL GUI builder - already there
- Clouseau - not yet

Use Eolian to do more

- The EFL GUI builder - already there
- Clouseau - not yet
- Ideas?

EFL interfaces

EFL interfaces

- Eoify more of the EFL

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win
  - `ecore_job` → event on the mailoop

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win
  - `ecore_job` → event on the mailoop
- Use advance Eo features

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win
  - `ecore_job` → event on the mailoop
- Use advance Eo features
  - Gesture layer API can be mostly trimmed (events tracking)

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win
  - `ecore_job` → event on the mailoop
- Use advance Eo features
  - Gesture layer API can be mostly trimmed (events tracking)
- Improve existing API

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win
  - `ecore_job` → event on the mailoop
- Use advance Eo features
  - Gesture layer API can be mostly trimmed (events tracking)
- Improve existing API
  - Common interfaces for highly redundant functions

EFL interfaces

- Eoify more of the EFL
  - `ecore_mainloop` → Eo object
  - `ecore_animator` → event on Elm_Win
  - `ecore_job` → event on the mailoop
- Use advance Eo features
  - Gesture layer API can be mostly trimmed (events tracking)
- Improve existing API
  - Common interfaces for highly redundant functions
  - Correct classification by inheritance

Documentation

Documentation

- 1st class citizen

Documentation

- 1st class citizen
- Support Eolian features

Documentation

- 1st class citizen
- Support Eolian features
- Write once, use everywhere

Documentation

- 1st class citizen
- Support Eolian features
- Write once, use everywhere
- Editable online?

Documentation

- 1st class citizen
- Support Eolian features
- Write once, use everywhere
- Editable online?

Documentation

- 1st class citizen
- Support Eolian features
- Write once, use everywhere
- Editable online? Comments like php.net?

Questions?

Tom Hacohen
tom.hacohen@samsung.com
http://stosb.com
@TomHacohen

Daniel Kolesa
d.kolesa@samsung.com
http://octaforge.org
@Octaforge

Resources Attributions

- Page **??**, resources/brewing_ifaces.png
- Page 2, resources/expert-beerpong.jpg
- Page 37, resources/one_syntax.jpg
- Page 66, resources/lua_beer.jpg
- Page 85, resources/python_cat.jpg
- Page 106, resources/pro_beer.jpg
- Page 107, resources/dessert.jpg