The EFL API in Review

stosb.com/talks

Tom Hacohen
Samsung Electronics Open Source Group
tom.hacohen@samsung.com
@TomHacohen

Today's Topics

Today's Topics

- Eo recap

Today's Topics

- Eo recap
- Interfaces recap

Today's Topics

- Eo recap
- Interfaces recap
- Major decisions for the future

Unify Code

Unify Code

- Many different object systems $\rightarrow$ one

Unify Code

- Many different object systems $\rightarrow$ one
- Many different event/callback implementations $\rightarrow$ one

Unify Code

- Many different object systems $\rightarrow$ one
- Many different event/callback implementations $\rightarrow$ one
- Make objects compatible

Reducing our API

Reducing our API

We have:

```
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);
```

Bindings Generation

Bindings Generation

- Be able to automatically generate for most popular languages

Bindings Generation

- Be able to automatically generate for most popular languages
- Correctly handle ref counting, buffer ownership and etc.

Not Hurt Performance

Not Hurt Performance

- Not easily measurable – many changes in EFL

Other Languages

Other Languages

- C++ – our developers hate it

Other Languages

- C++ – our developers hate it
- Objective C – quite ugly and not really common in OSS world

Other Languages

- C++ – our developers hate it
- Objective C – quite ugly and not really common in OSS world
  - We considered using just the runtime

GObject

GObject

Good:

- Fast

GObject

Good:

- Fast
- Has a “C feel”

GObject

Good:

- Fast
- Has a “C feel”

GObject

Good:

- Fast
- Has a “C feel”

Bad:

- Doesn’t offer a stable ABI

GObject

Good:

- Fast
- Has a “C feel”

Bad:

- Doesn’t offer a stable ABI
- Funny, full of casting syntax

GObject

Good:

- Fast
- Has a “C feel”

Bad:

- Doesn’t offer a stable ABI
- Funny, full of casting syntax
- “G tech” dependencies

GObject

Good:

- Fast
- Has a "C feel"

Bad:

- Doesn't offer a stable ABI
- Funny, full of casting syntax
- "G tech" dependencies
- Didn't exactly fit our needs

Basics

Basics

- It's Enlightenment's (fairly) new object system

Basics

- It's Enlightenment's (fairly) new object system
- Supports classes, abstract classes, mixins and interfaces

Basics

- It's Enlightenment's (fairly) new object system
- Supports classes, abstract classes, mixins and interfaces
- Completely written in C (no external preprocessor)

Basics

- It's Enlightenment's (fairly) new object system
- Supports classes, abstract classes, mixins and interfaces
- Completely written in C (no external preprocessor)
- API/ABI stable

Basics

- It's Enlightenment's (fairly) new object system
- Supports classes, abstract classes, mixins and interfaces
- Completely written in C (no external preprocessor)
- API/ABI stable
- Portable

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_ret(obj, tmp, elm_widget_enabled_get()))`

Using Eo

- `eo_do(obj, efl_file_set("file.eet", "key"));`
- `if (eo_do_ret(obj, tmp, 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_ret(obj, tmp, 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_ret(obj, tmp, 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));
  ```
  ```

`eo_do()` – How It's Done (simplified)

eo_do() – How It's Done (simplified)

```
#define eo_do(eoid, ...) \
do {                                \
    _eo_do_start(eoid);  \
    __VA_ARGS__;                    \
    _eo_do_end();                   \
} while (0)
```

`eo_do_ret()` – How It's Done (simplified)

eo_do_ret() – How It's Done (simplified)

```
#define eo_do_ret(eoid, ret_tmp, func) \
(                                      \
    _eo_do_start(eoid),                \
    ret_tmp = func,                    \
    _eo_do_end(),                      \
    ret_tmp                            \
)
```

Defining New Functions (simplified)

Defining New Functions (simplified)

```
EOAPI EO_FUNC_BODY(eo_parent_get, Eo *, NULL);
```

Defining New Functions (simplified)

```
EOAPI EO_FUNC_BODY(eo_parent_get, Eo *, NULL);
```

```
#define EO_FUNC_BODY(Name, Ret, DefRet)                \
Ret Name(void)                                         \
{                                                      \
    static Eo_Op op = EO_NOOP;                         \
    if (op == EO_NOOP)                                 \
        op = _eo_api_op_id_get((void*) Name);          \
    if (!_eo_call_resolve(#Name, op, &call))           \
        return DefRet;                                 \
    _Eo_##Name##_func _func_ =                         \
        (_Eo_##Name##_func) call.func;                 \
    return _func_(call.obj, call.data);                \
}
```

Defining New Classes (simplified)

Populating a struct with some metadata

Defining New Classes (simplified)

Populating a struct with some metadata

```
static Eo_Op_Description _edje_object_op_desc[] = {
    EO_OP_FUNC(edje_obj_update_hints_set, ↩
        _edje_object_update_hints_set),
    EO_OP_FUNC_OVERRIDE(eo_constructor, ↩
        _edje_object_eo_base_constructor),
    EO_OP_CLASS_FUNC(eo_event_global_thaw, ↩
        _eo_base_event_global_thaw),
    EO_OP_CLASS_OVERRIDE_FUNC(eo_event_global_thaw, ↩
        _edje_object_eo_base_event_global_thaw)
};
```

Event Identifiers

Event Identifiers

```
EOAPI const Eo_Event_Description ↩
    _EO_BASE_EVENT_CALLBACK_ADD = ↩
    EO_EVENT_DESCRIPTION("callback,add");
```

Unique Features

Unique Features

- Pointer indirection (at least in C)

Unique Features

- Pointer indirection (at least in C)
- Multiple calls in one context

Unique Features

- Pointer indirection (at least in C)
- Multiple calls in one context
- How we do constructors (setting properties, no constructors)

Unique Features

- Pointer indirection (at least in C)
- Multiple calls in one context
- How we do constructors (setting properties, no constructors)
- Named ref-counting

Unique Features

- Pointer indirection (at least in C)
- Multiple calls in one context
- How we do constructors (setting properties, no constructors)
- Named ref-counting
- Composite objects

Unique Features

- Pointer indirection (at least in C)
- Multiple calls in one context
- How we do constructors (setting properties, no constructors)
- Named ref-counting
- Composite objects
- Default return values

Wash, Rinse, Repeat

Wash, Rinse, Repeat

- Eo1

Wash, Rinse, Repeat

- Eo1
- Eo2

Wash, Rinse, Repeat

- Eo1
- Eo2
- Eolian

Wash, Rinse, Repeat

- Eo1
- Eo2
- Eolian
- Eolian (improved)

Stability

Stability

- Pointer indirection saved us in many cases

Stability

- Pointer indirection saved us in many cases
- We caught a lot of errors that were not noticed before

Stability

- Pointer indirection saved us in many cases
- We caught a lot of errors that were not noticed before
- Single point of access for type checking makes it impossible to forget

Reduced API

Reduced API

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);
```

Reduced API

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);
```

Now:

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

eo_del(obj);
```

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 → easy bindings

A new format?

- Language independent → easy bindings
- Familiar syntax → easy to pick up

A new format?

- Language independent → easy bindings
- Familiar syntax → easy to pick up
- Easy to read and write

A new format?

- Language independent → easy bindings
- Familiar syntax → 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 (under development): JavaScript, Python, Rust and OCaml

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 are still missing

However...

- Some things are still missing
- Documentation?

However...

- Some things are still missing
- Documentation?
- Value ownership

However...

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

However...

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

However...

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

And yet...

However...

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

And yet...

- Very useful

However...

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

And yet...

- Very useful
- Generic

However...

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

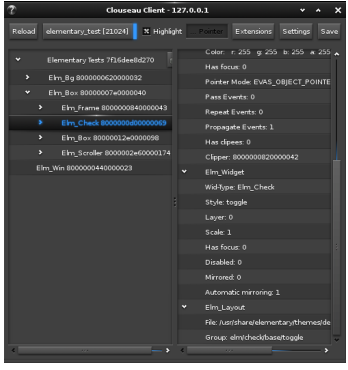
And yet...

- Very useful
- Generic
- I'd like to get it adopted by others (non EFL)

Clouseau

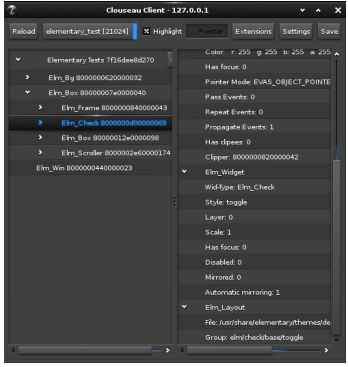
Clouseau

- Is there anyone who doesn't know this one by now?



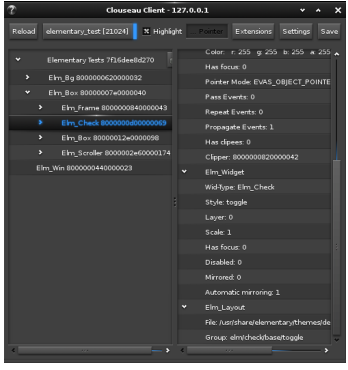
Clouseau

- Is there anyone who doesn't know this one by now?
- Application state inspector for the EFL



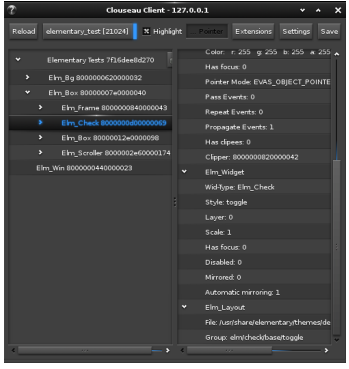
Clouseau

- Is there anyone who doesn't know this one by now?
- Application state inspector for the EFL
- Was not created following Eo (but greatly improved)



Clouseau

- Is there anyone who doesn't know this one by now?
- Application state inspector for the EFL
- Was not created following Eo (but greatly improved)
- Will get even better with Eolian



Erigo

Erigo

- EFL GUI builder

Erigo

- EFL GUI builder
- Reads properties from Eolian

Erigo

- EFL GUI builder
- Reads properties from Eolian
  - Supports whatever version is installed on the system automatically

Erigo

- EFL GUI builder
- Reads properties from Eolian
  - Supports whatever version is installed on the system automatically
  - Supports widgets that it has no notion of

Erigo

- EFL GUI builder
- Reads properties from Eolian
  - Supports whatever version is installed on the system automatically
  - Supports widgets that it has no notion of
- Has it's own format that is processed by language specific code generators

Espion

Espion

- Goal: easily import GUIs to Erigo

Espion

- Goal: easily import GUIs to Erigo
- Intercepts `eo_add()` and `eo_do()`

Espion

- Goal: easily import GUIs to Erigo
- Intercepts `eo_add()` and `eo_do()`
- Uses Eolian to correctly process the calls

What is it about (again)?

What is it about (again)?

- Fixing up the EFL API and inheritance

What is it about (again)?

- Fixing up the EFL API and inheritance
- Utilising the new Eo features

What is it about (again)?

- Fixing up the EFL API and inheritance
- Utilising the new Eo features
- Annotating the EFL API for generated bindings

What is it about (again)?

- Fixing up the EFL API and inheritance
- Utilising the new Eo features
- Annotating the EFL API for generated bindings
- Creating new classes that are important for the life cycle

What is it about (again)?

- Fixing up the EFL API and inheritance
- Utilising the new Eo features
- Annotating the EFL API for generated bindings
- Creating new classes that are important for the life cycle
- “Insanely important” — Carsten Haitzler (A few hours ago)

What is it about (again)?

- Fixing up the EFL API and inheritance
- Utilising the new Eo features
- Annotating the EFL API for generated bindings
- Creating new classes that are important for the life cycle
- "Insanely important" — Carsten Haitzler (A few hours ago)
- Everything that's even remotely related to Eo and EFL API and is dumped upon this task arbitrarily.

Fixing API and Inheritance

Fixing API and Inheritance

- Making all of the Ecore_* Eo objects

Fixing API and Inheritance

- Making all of the Ecore_* Eo objects
- Removing duplicated APIs

```
elm_layout_part_text_set()
edje_object_part_text_set()
elm_object_text_set()
```

Fixing API and Inheritance

- Making all of the Ecore_* Eo objects
- Removing duplicated APIs

```
elm_layout_part_text_set()
edje_object_part_text_set()
elm_object_text_set()
```

- Make `Elm.Layout` implement `Edje.Object`

Using Eo Features

Using Eo Features

- Moving Ecore.Animator to be a signal on the window

Using Eo Features

- Moving Ecore.Animator to be a signal on the window
- Moving Ecore.Job to be a signal on the mainloop

Annotating for Bindings

Annotating for Bindings

- Add `@own` to relevant parameters

Annotating for Bindings

- Add `@own` to relevant parameters
- Add enums/struct definitions in .eo files (when public)

Annotating for Bindings

- Add `@own` to relevant parameters
- Add enums/struct definitions in .eo files (when public)
- Use correct class names instead of `Eo *`

New Classes

New Classes

- Mainloop object

New Classes

- Mainloop object
- Application object

Questions so far?

Nullability

Nullability

- Change and split @notnull to @nullable and @optional

Nullability

- Change and split @notnull to @nullable and @optional
- Very useful for languages that support this notion (e.g. Rust and C++)

Nullability

- Change and split @notnull to @nullable and @optional
- Very useful for languages that support this notion (e.g. Rust and C++)
- Leads to safer code and more information about types

Nullability

- Change and split @notnull to @nullable and @optional
- Very useful for languages that support this notion (e.g. Rust and C++)
- Leads to safer code and more information about types
- Side effect: stop using `EINA_ARG_NONNULL`

Thread Safety

Thread Safety

- Eo infra is thread safe, objects aren't

Thread Safety

- Eo infra is thread safe, objects aren't
- Is there a useful case which requires we change that?

Shared Interfaces

Shared Interfaces

- Share most of the EFL's functions

Shared Interfaces

- Share most of the EFL's functions
- Change most functions to the EFL interfaces

Shared Interfaces

- Share most of the EFL's functions
- Change most functions to the EFL interfaces
- Forces API to be consistent

Shared Interfaces

- Share most of the EFL's functions
- Change most functions to the EFL interfaces
- Forces API to be consistent
- Limits the flexibility of API (because everything is shared)

Shared Interfaces

- Share most of the EFL's functions
- Change most functions to the EFL interfaces
- Forces API to be consistent
- Limits the flexibility of API (because everything is shared)
- Can cause clashes (parent class uses a function for one thing, child for another)

Short Names

Short Names

- `efl_file_set()`, `efl_color_set()` ...

Short Names

- `efl_file_set()`, `efl_color_set()` ...
- C API matches OOP languages' API (C++, JS, Lua)

Short Names

- `efl_file_set()`, `efl_color_set()` ...
- C API matches OOP languages' API (C++, JS, Lua)
- Maintain the long names (full namespacing)?

Short Names

- `efl_file_set()`, `efl_color_set()` ...
- C API matches OOP languages' API (C++, JS, Lua)
- Maintain the long names (full namespacing)?
- Developers seem to prefer short names (though come from OOP background)

Short Names

- `efl_file_set()`, `efl_color_set()` ...
- C API matches OOP languages' API (C++, JS, Lua)
- Maintain the long names (full namespacing)?
- Developers seem to prefer short names (though come from OOP background)
- Improves API consistency (same name does the same)

Short Names

- `efl_file_set()`, `efl_color_set()` ...
- C API matches OOP languages' API (C++, JS, Lua)
- Maintain the long names (full namespacing)?
- Developers seem to prefer short names (though come from OOP background)
- Improves API consistency (same name does the same)
- Have a generation pass across all eo files in efl (second stage?)

Short Names

- `efl_file_set()`, `efl_color_set()` ...
- C API matches OOP languages' API (C++, JS, Lua)
- Maintain the long names (full namespacing)?
- Developers seem to prefer short names (though come from OOP background)
- Improves API consistency (same name does the same)
- Have a generation pass across all eo files in efl (second stage?)
- Detect conflicts

Small vs. Sparse Classes

Small vs. Sparse Classes

- Small classes that have to be fully implemented

Small vs. Sparse Classes

- Small classes that have to be fully implemented
- Big classes with many optional properties

Small vs. Sparse Classes

- Small classes that have to be fully implemented
- Big classes with many optional properties
- Which do we want?

Questions or comments?

- Nothing