Canvas EO APIs

New EFL interfaces
Images, Containers and Window APIs cleanup

EFL developers days 2016
Jean-Philippe ANDRE
2016/05/14
Paris

Scope

- Images
- Windows
- Containers
- Lunch

Canvas Images

- Legacy:
  - evas_object_image_[filled_]add()
    - File images
    - Proxies with source object
    - GL view (Evas GL)
    - 3d view (Evas 3d)
    - Snapshot objects
    - Native surfaces (Xpixmap, EGLImage, …)

Canvas Image

- One object that does everything
  - Huge number of APIs (82 + object apis)
  - Complex internal code
    - Some of the genericity is good, some is excessive
  - Various “types” of image objects
    - Many APIs valid for a certain type only
    - Potentially confusing to use

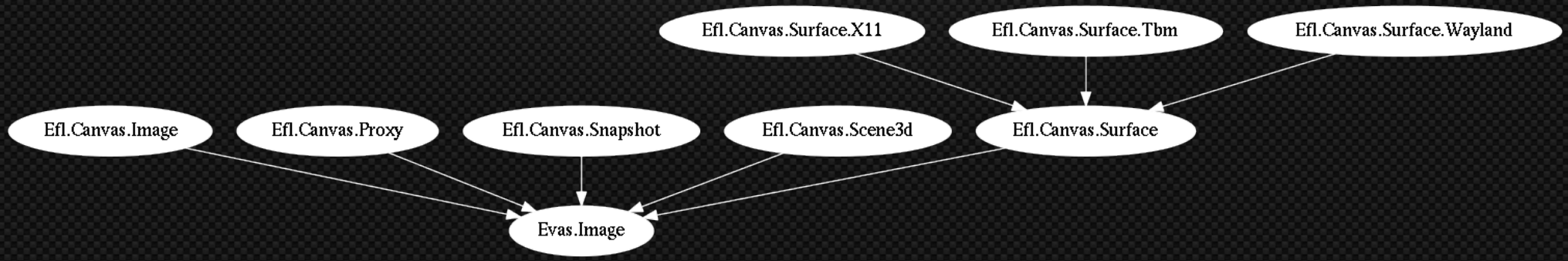
Canvas Image

- Why splitting?
  - *Potentially* improving the internal code
  - Common set of interfaces for all images
    - But remove excess fluff
  - Simplify the API set for each image type
    - Eg. snapshot has no custom method

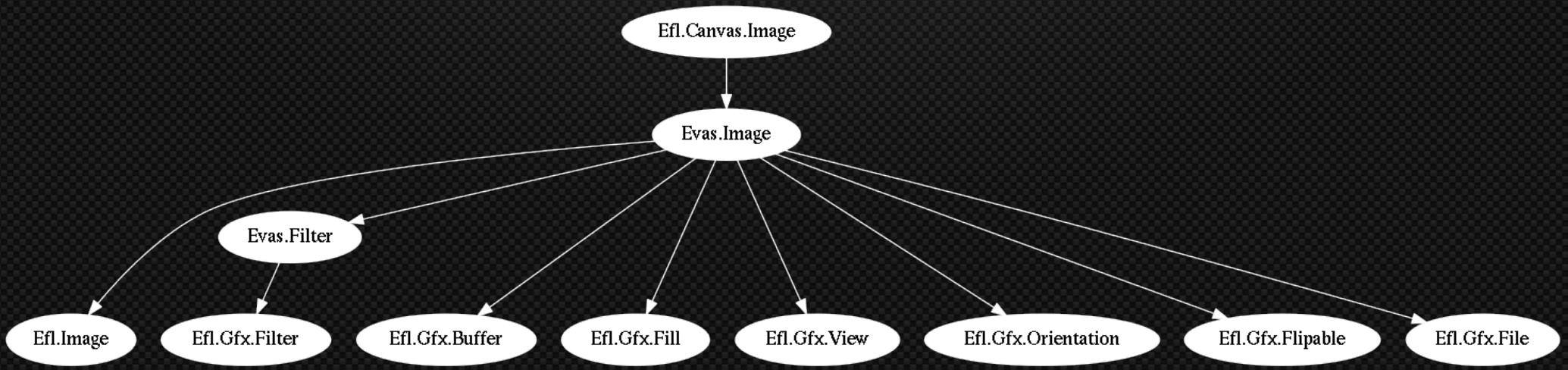
Canvas Image

- Evas.Image (internal)
  - Efl.Canvas.Image
    - file_set
  - Efl.Canvas.Proxy
  - Efl.Canvas.Scene3d
  - Efl.Canvas.Snapshot
  - Efl.Canvas.Surface.{Tbm,X11,Wayland}
    - Native surfaces, limited to C/C++

Canvas Image



Canvas Image



Canvas Image

- Legacy data_set/get
  - Dual meaning:
    - External pointer data
    - Get internal pixel data
      - Call set to release pointer
- Map/unmap
  - convert colorspaces
  - region mapping
  - proxy image contents, GL view content, etc…
    - Direct GL map or glReadPixels
  - buffer_data_set for external data

Canvas Image

- Remaining issues (TODO):
  - map/unmap not implemented in some cases:
    - GL engine (TODO)
    - YUV & other planar formats (need API redesign)
  - async file set API (TODO)
    - Eina Promise
    - *Fake it till you make it*
  - Use & test the new APIs
    - Legacy still works, apparently...
  - Anything else?

UI Image

- Efl.Ui.Image
  - Elementary Image
  - Icon is a property of Efl.Ui.Image
  - Thumb & Photo are legacy-only (aka. Dead)
  - New scaling API

UI Image

- New scaling API
  - Center, fit_inside, fit_outside, fill
  - Scale up, down

Interlude #1

Windows

- Merge all canvases:
  - evas
  - ecore_evas
  - elm_win
- All work together and share APIs
  - Unify under a single Window class

Windows

- Merging
  - No actual code merge
    - Different libs, lots of complex & fragile code
  - Add missing APIs from Evas to Window

Windows

- API merge
  - Evas
  - Ecore Evas
  - Elementary Window
  - Conformant
    - Indicator, quickpanel, virtual keyboard
  - IMF (todo)
- Input events redesign

Windows

- Questions
  - Which parts?
    - Keep stack model (resize_object_add)
    - “bg” + “content” only

Windows

- Naviframe is dead, long live… uh… windows
  - Naviframe items → windows
  - Compositor handles stacking
    - Multiple apps in same stack
- Elm Flip becomes linear container
  - In-app view stack
  - Can replace naviframe

Input Events

- Origins and targets:
  - Ecore (X, drm, ...)
  - Evas
  - Evas Object
  - Elm Gesture
- Different mechanisms
  - Ecore events, Function calls, Evas events, ...
- Different event info structs

Input Events

- Unify all events types with EO
- EO events everywhere
  - Ecore → Evas → Objects
- Event info data is an EO
  - Extensibility
  - Raw & high level properties
  - Private event data shared across EFL internals

Input Events

- Raw events
  - Mouse move/down/up/in/out, wheel, etc...
  - Resampled values preferred
    - Raw timestamps & position available
  - Event origin & device
- High level events
  - Clicks, gestures

Input Events

- Reality hits
  - Legacy behaviour unmodifiable
  - Pointer event vs. mouse move, down, up, ...
  - Use legacy functions to send events
    - Add “reserved” field to carry EO data
- Current (W.I.P.)
  - Ecore → (event) → Ecore Input Evas → (func) → Ecore Evas → (eo_event) → Evas → (legacy evas event) → Evas Object
  - Evas event info ↔ Eo pointer event

Input Events

- Questions / TBD
  - Unify all events?
    - Only one pointer event
  - Keep similar behaviour?
    - Mouse move/down/up, multi move/down/up, …
    - Only change event_info to be Pointer Event data
  - Gestures & high level events?
  - Resampling & smoothing
    - input vs. screen refresh rates

Interlude #2

Containers

- Box
- Table
- Grid
- Edje & Elm Layout
  - Edje Box & Edje Table
- Widgets, window, …

- ***Not*** for genlist (Efl.Model based)
  - Maybe some APIs can be covered

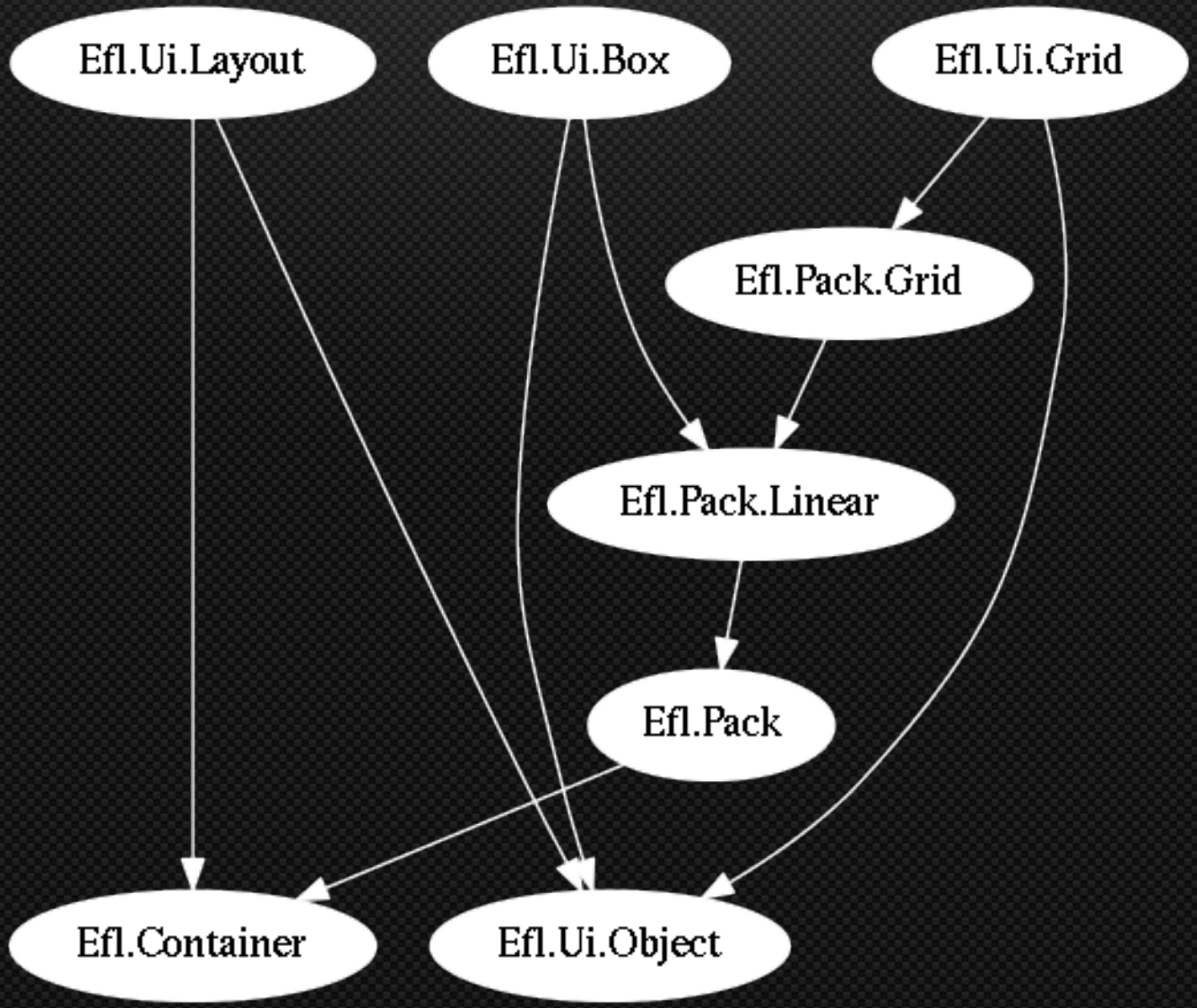
Pack API

- Unify all container APIs
- Contents
  - Named parts
  - Elements
- Containers
  - Slots: parts
  - Linear: 1d
  - Grid: 2d (3d)

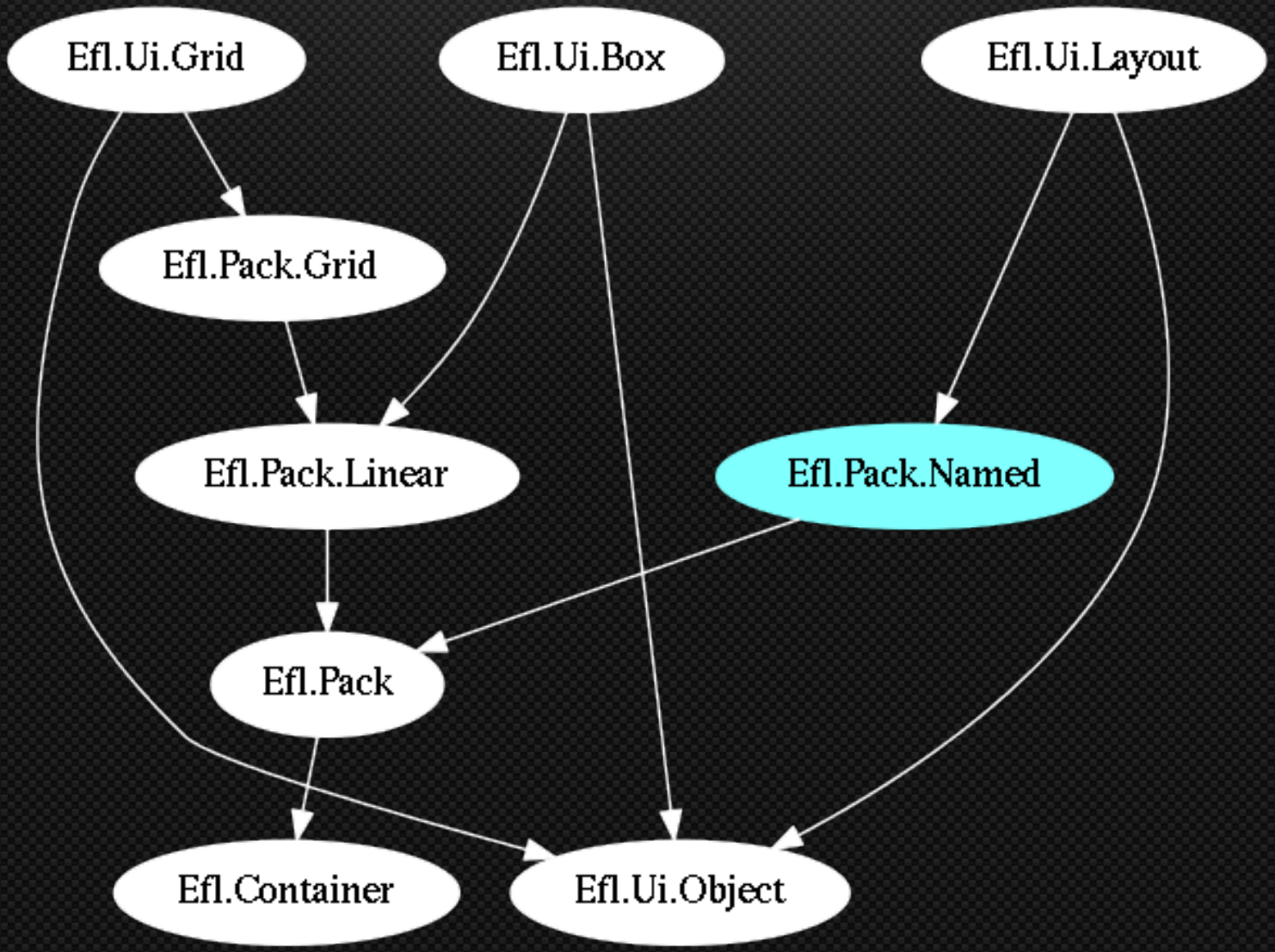
Pack API

- New features
  - Requested size hint (vs. content min hint)
    - TODO: proper EO API
  - Custom table/grid layout functions (& box)
  - Linear append in tables

Pack API


Efl.Ui.Layout
Efl.Ui.Box
Efl.Ui.Grid
Efl.Pack.Grid
Efl.Pack.Linear
Efl.Pack
Efl.Container
Efl.Ui.Object

Pack API (proposal)



Pack API

- Custom Layouts
  - Class inherit
    - → Implement layout func
  - Layout Engine
    - → @class function
    - Used as poor man's function pointer
      - On-the-fly set
      - Or object class (eg. Efl.Ui.Box.Flow)

Interlude #3

Controversy

- Not yet time for lunch!
- Part API debacle
  - Review of alternatives
  - Conclude on preferred solution

EFL Part APIs

- 1. Part argument
  - @optional
    - Last arg
  - C macros

- Pros
  - Straighforward
  - Good for bindings

- Cons
  - “Leaks” part name in too many APIs
  - Where is part needed?

EFL Part APIs

- 2. eo_part
  - Core EO feature
  - Same as eo_super
- Pros
  - Core feature
  - No object leak
  - No API leak
- Cons
  - Extra bit / bit reuse
  - Manual mapping to bindings
  - Not a real object
  - Locks

EFL Part APIs

- 3. Get real object
  - Evas Text, Box, …

- Pros
  - Same functions as real object
  - Out of the box
- Cons
  - Direct access to parts
  - No API control or stability
  - Evas objects are internals

EFL Part APIs

- 4. Proxy object
  - EO object
  - Part name and container EO
  - Created on the fly
- Pros
  - No API leak
  - Restricted API set
- Cons
  - None, obviously

EFL Part APIs

- @jpeg's implementation:
  - Edje & Elm Layout: BOX + TABLE
  - Uses a proxy object
    - Created on the fly in efl_content_get
    - References part name (string) and container (Eo*)
    - Never dies
    - Manual implementation of proxy functions
    - eo_unref / eo_del kills it
      - Wait, what?

EFL Part APIs

- efl_pack(efl_content_get(layout, “box”), obj);

- Eo *box = efl_content_get(layout, “box”);
- efl_pack(box, obj1);
- efl_pack(box, obj2);
- eo_unref(box);

EFL Part APIs

```
Eo *box = efl_content_get(layout, “box”);
efl_pack(box, obj1);
trouble(layout);
efl_pack(box, obj2);
eo_unref(box);

trouble (layout) {
    Eo *box = efl_content_get(layout, “box”);
    eo_unref(box);
}
```

EFL Part APIs

- 5. @tasn's proposal
  - Same proxy object
  - Temporary object
  - Valid for 1 call
    - Implicit unref
- Pros
  - Same as above
  - Clear lifecycle
- Cons
  - EO object (perf?)
  - Extra work for @jpeg

EFL Part APIs

- efl_pack(efl_part(layout, “box”), obj);

- Eo *box = eo_ref(efl_part(layout, “box”));
- efl_pack(box, obj1);
- efl_pack(box, obj2);
- eo_unref(box);

Time for lunch