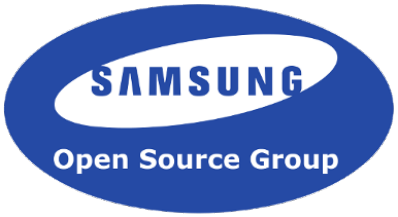
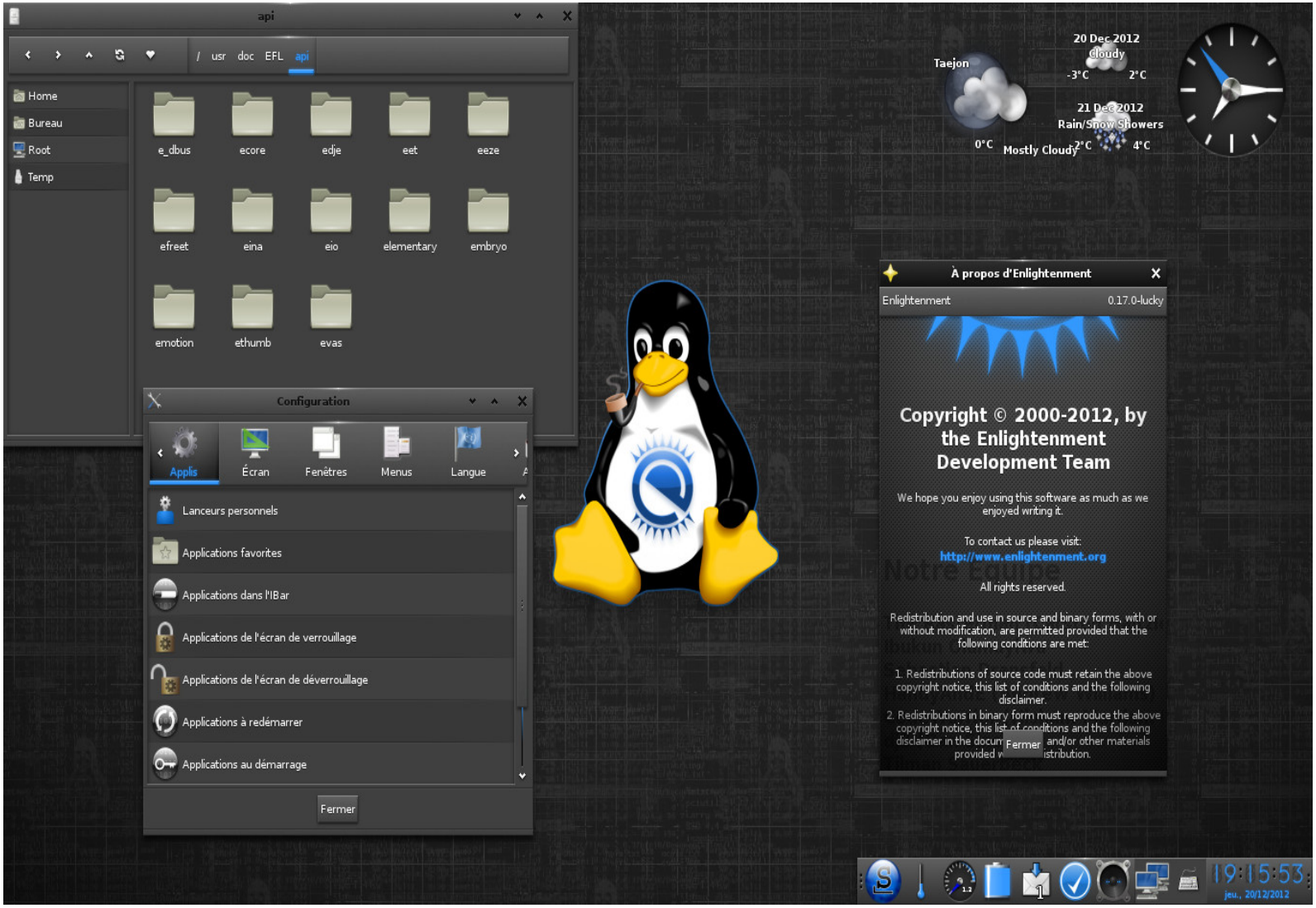


Enlightenment Foundation Libraries 2.0

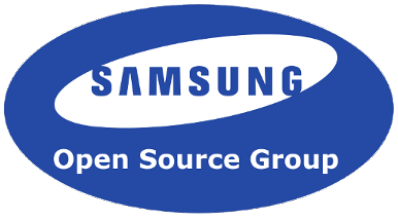
Time to rethink and make things easier !

Cedric BAIL
Samsung Open Source Group
cedric@osg.samsung.com

EFL: A Toolkit Created for Enlightenment 17

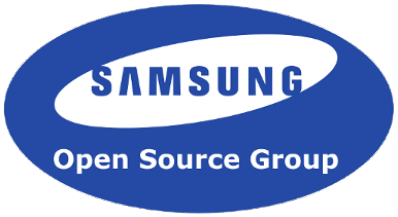


Enlightenment 17



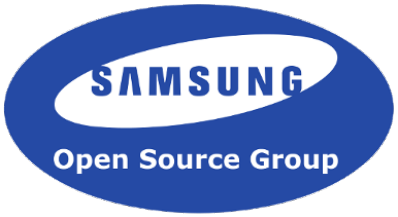
- Enlightenment project started in 1997
- Windows Manager
- First Windows Manager of GNOME
- Full rewrite started in 2001
- Primary belief is there will never be *“a year of the Linux desktop”*
- Designed with the embedded world in mind…
- … and needed a toolkit !
- As none matched our need back then and still don’t today !

Enlightenment Foundation Libraries (*EFL*)



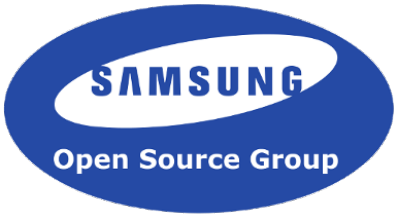
- Spent a decade writing a modern graphic toolkit
- Licensed under a mix of LGPL 2.1 and BSD license
  (Yes, nobody own it, nobody can change the license)
- Focus on embedded devices
- Used in Samsung Tizen product
- First release on January 2011
- Stable, long term API/ABI
- In the process of releasing version 1.21

State of EFL



- Designed for creating a Windows Manager (WM), now used for any type of application
- Has its own scene graph and rendering library
- Optimized to reduce CPU, GPU, memory and battery usage
- Supports international language requirements (LTR/RTL, UTF8)
- Supports all variations of screens and input devices (scale factor)
- Supports accesibility (ATSPI)
- Fully Themable (layout of the application included)
- Supports profiles
- Can take up as little as 8MB of space with a minimal set of dependencies
- Has a modular design

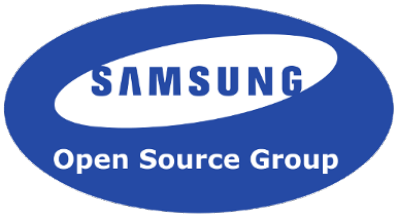
State of EFL



- More than 15 years of organic grow
- Always focused on performance
- Little has been done on our API

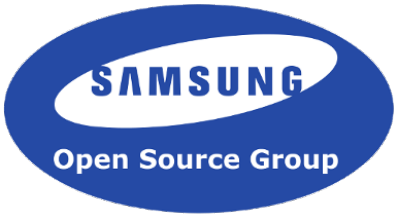
Road to EFL 2.0

What is the goal ?



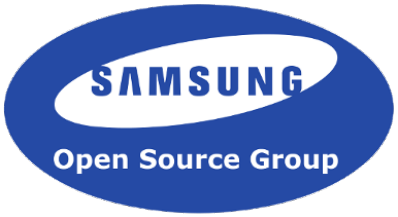
- Easier to maintain bindings !
  - Everyone has their preferred language
  - Thousand of API to port for each language
  - Documentation has to be provided too
  - Lots of work that needs to be done for every release...

What is the goal ?



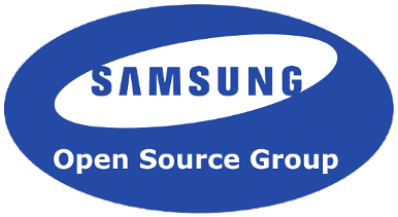
- Simpler API
  - One object model
  - One event system
  - One asynchronous system
  - Refactor functions to do the same thing on every object
- Modern paradigm
  - Object lifecycle
  - Asynchronous chains

What is the goal ?



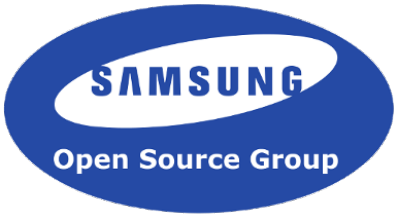
- As it will take time to roll out
  - → API/ABI compatibility has to be maintained
- Possibility to slowly migrate code
  - → Use the new API with old API

What is the goal ?



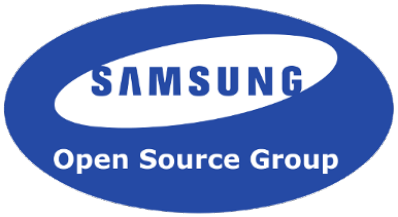
- Preserve
  - Energy efficiency of CPU/GPU usage
  - Memory usage
  - Scalability

Current progress



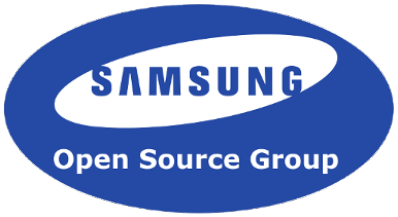
- Object model
  - Refcounting, no auto-del (binding happy)
  - Parent refcounting hability
  - Classic simple lifecycle
  - Events
  - Eolian language for generating
    - C boiler plate
    - All bindings !
    - Documentation

Eolian



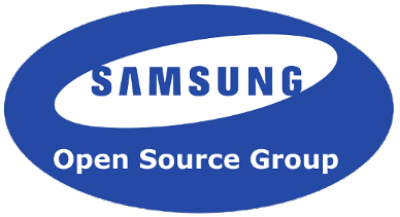

```
abstract Efl.Object ()
{
   [[Abstract Efl object class]]
   methods {
      @property parent {
         [[The parent of an object.]]
         set {}
         get {}
         values {
            parent: Efl.Object @nullable; [[The new parent]]
         }
      }
      del @const {
         [[Unrefs the object and reparents it to NULL]]
      }
   }
}
```

Current progress



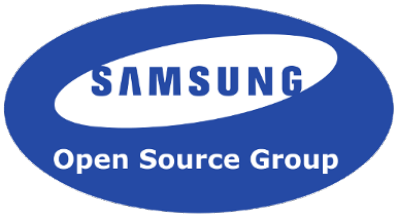
- Application can be of 3 types
  - Command line → Efl_Core.h
  - Network → Efl_Net.h
  - User interface → Efl_UI.h
- Auto initialisation of the relevant component
- Provide quicklaunch support

Current progress



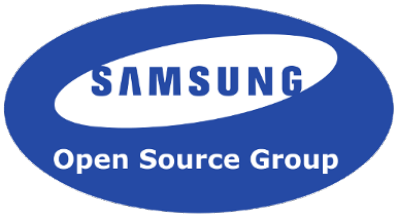
- Application main object is the main loop
  - Provide application lifecycle (pause, resume, terminate)
  - Provide activity events (idle, job)
  - Build information (efl version and application build version)
- Canvas use a main loop to drive rendering
- Object use the canvas to drive animation

Current progress - Asynchronous



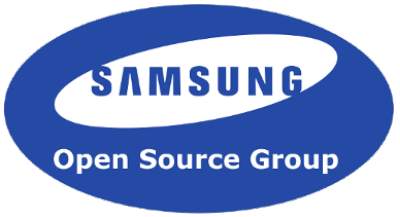
- Promise/Future
  - Like a pipe that will deliver one value guaranteed
  - Promise → Write side/API developer
  - Future → Read side/API user
- Close to C++ primitive
- Allow for chaining asynchronous operation
- Synchronisation primitive (wait for all, race)

Current progress - Network



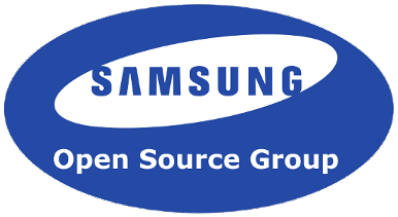
- One abstraction for every protocol
- Even stdin/stdout can be abstracted with it
- Allow for easy chaining logic

Current progress – Graphics



- Multiple meaningful namespace :
  - Efl.Canvas → 2D Graphics primitive
  - Efl.CanvasVG → Vector graphics primitive
  - Efl.Canvas3D → 3D graphics primitive
  - Efl.Ui → User interface widgets
- Refactor API and make it compliant with bindings constraint

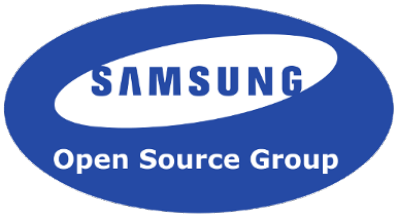
Current progress – Graphics



- Efl.Canvas is mostly done.
- Focus is now on making Efl.Ui a widgets set useful/working for mobile/touchscreen/tv
- Later release will improve desktop support

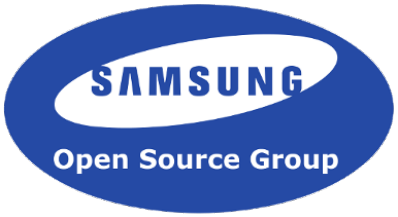
Work ahead

On going



- Clear up the thread model, long on going discussion
- Model View ViewModel infrastructure
  - Just connect a widget property to a model property
  - Fully asynchronous
  - Simplify testing
  - Increase reusability

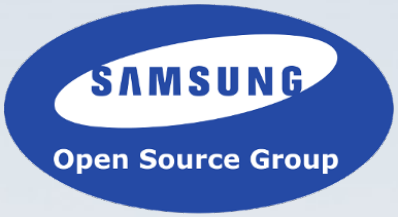
On going



- Improve documentation/tutorials
- Support more languages automatically (Python, C#, Lua, JS, go, ...)
- Improve automatic testing of the API (unit testing and integration testing)
- Feature parity on the widgets set with legacy

SAMSUNG
Open Source Group

Questions ?



Thank You!