Cuite: Qt 5 bindings for OCaml

Frédéric Bour

November 8, 2017

Qt?

- A C++ framework for Desktop and Mobile apps

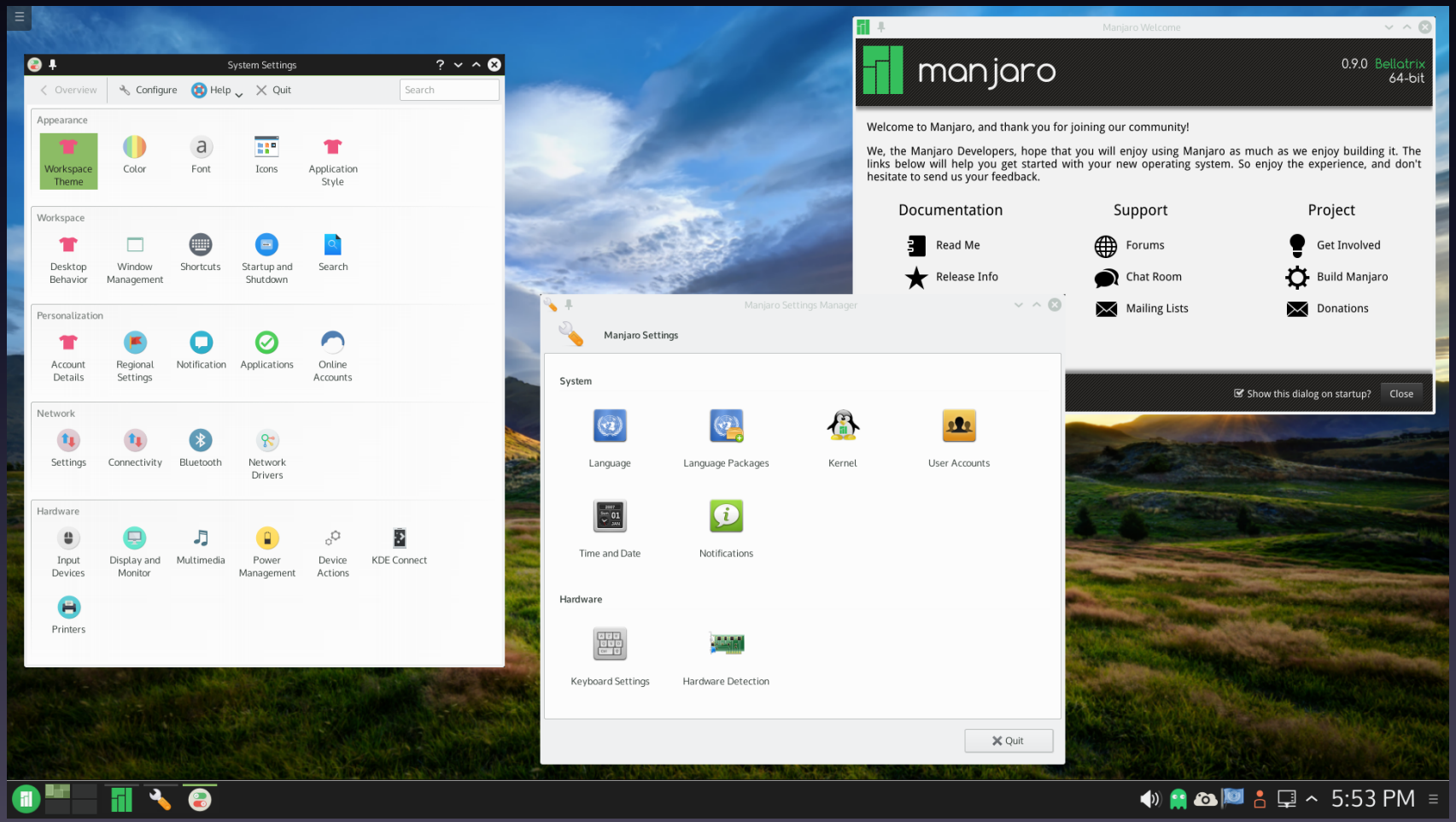
Qt?

- A C++ framework for Desktop and Mobile apps
- Abstracting "platforms":
  (stdlib, **windowing system**, **GUI**, VFS, networking, "XML", …)

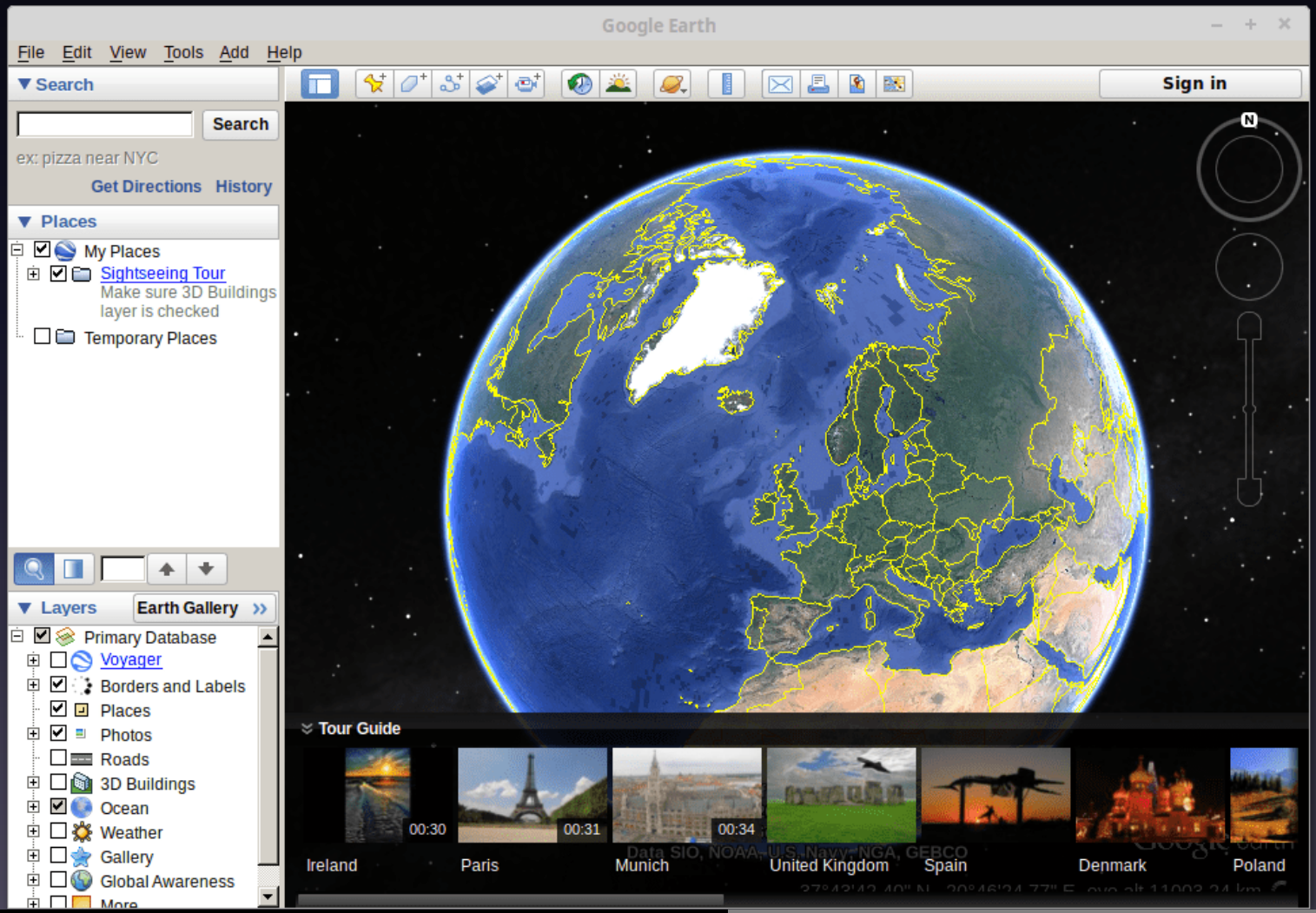
Qt?

- A C++ framework for Desktop and Mobile apps
- Abstracting "platforms":
  (stdlib, **windowing system**, **GUI**, VFS, networking,
  "XML", …)
- Cross-platform:
  Linux (X11, Wayland, Android), macOS, iOS, Windows,
  Blackberry, Sailfish OS, …

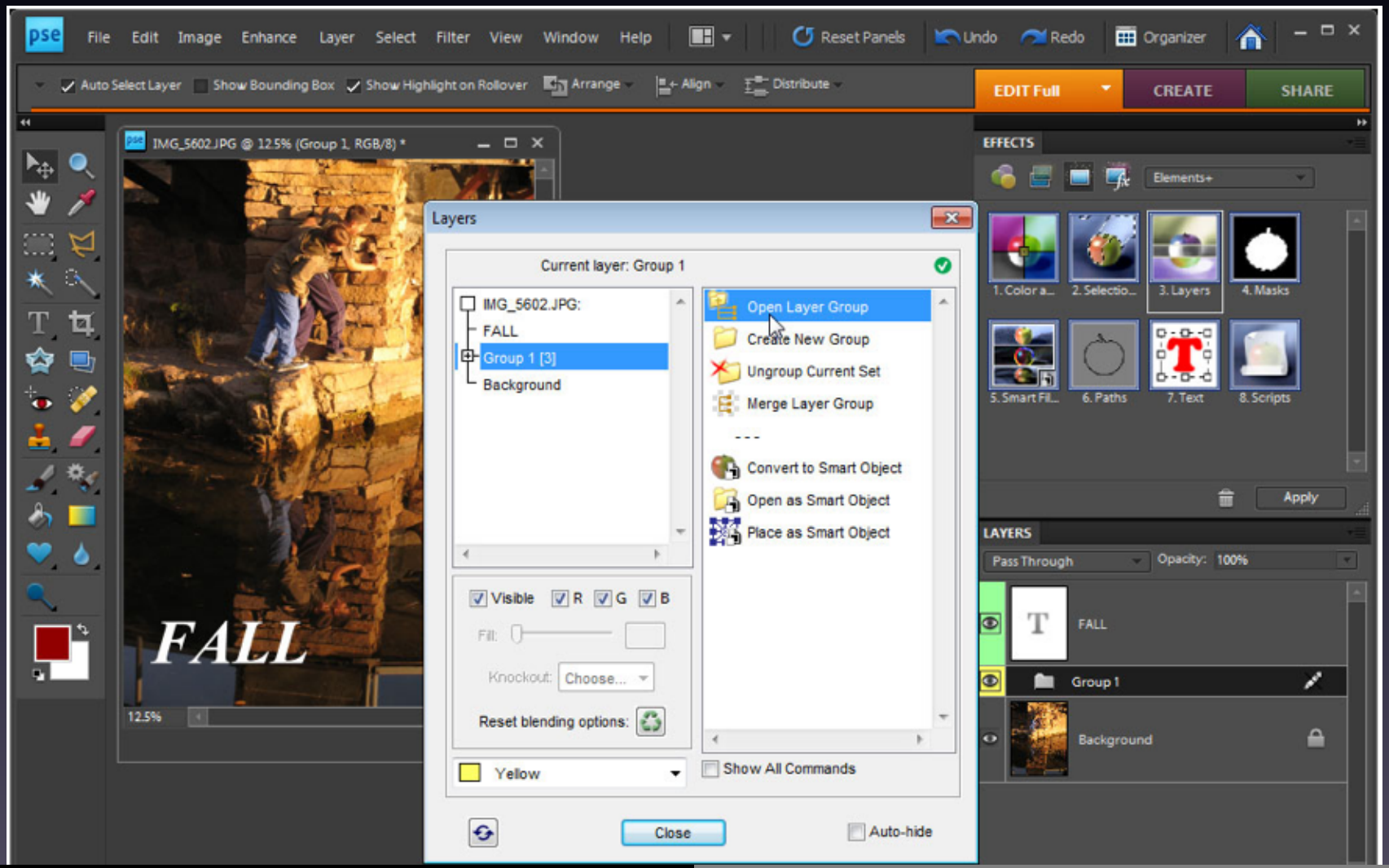
Used by KDE


System Settings
Overview
Configure
Help
Quit
Search
Appearance
Workspace Theme
Color
Font
Icons
Application Style
Workspace
Desktop Behavior
Window Management
Shortcuts
Startup and Shutdown
Search
Personalization
Account Details
Regional Settings
Notification
Applications
Online Accounts
Network
Settings
Connectivity
Bluetooth
Network Drivers
Hardware
Input Devices
Display and Monitor
Multimedia
Power Management
Device Actions
KDE Connect
Printers
Manjaro Welcome
manjaro
0.9.0 Bellatrix
64-bit
Welcome to Manjaro, and thank you for joining our community!
We, the Manjaro Developers, hope that you will enjoy using Manjaro as much as we enjoy building it. The links below will help you get started with your new operating system. So enjoy the experience, and don't hesitate to send us your feedback.
Documentation
Read Me
Release Info
Support
Forums
Chat Room
Mailing Lists
Project
Get Involved
Build Manjaro
Donations
Show this dialog on startup?
Close
Manjaro Settings Manager
Manjaro Settings
System
Language
Language Packages
Kernel
User Accounts
Time and Date
Notifications
Hardware
Keyboard Settings
Hardware Detection
Quit
5:53 PM

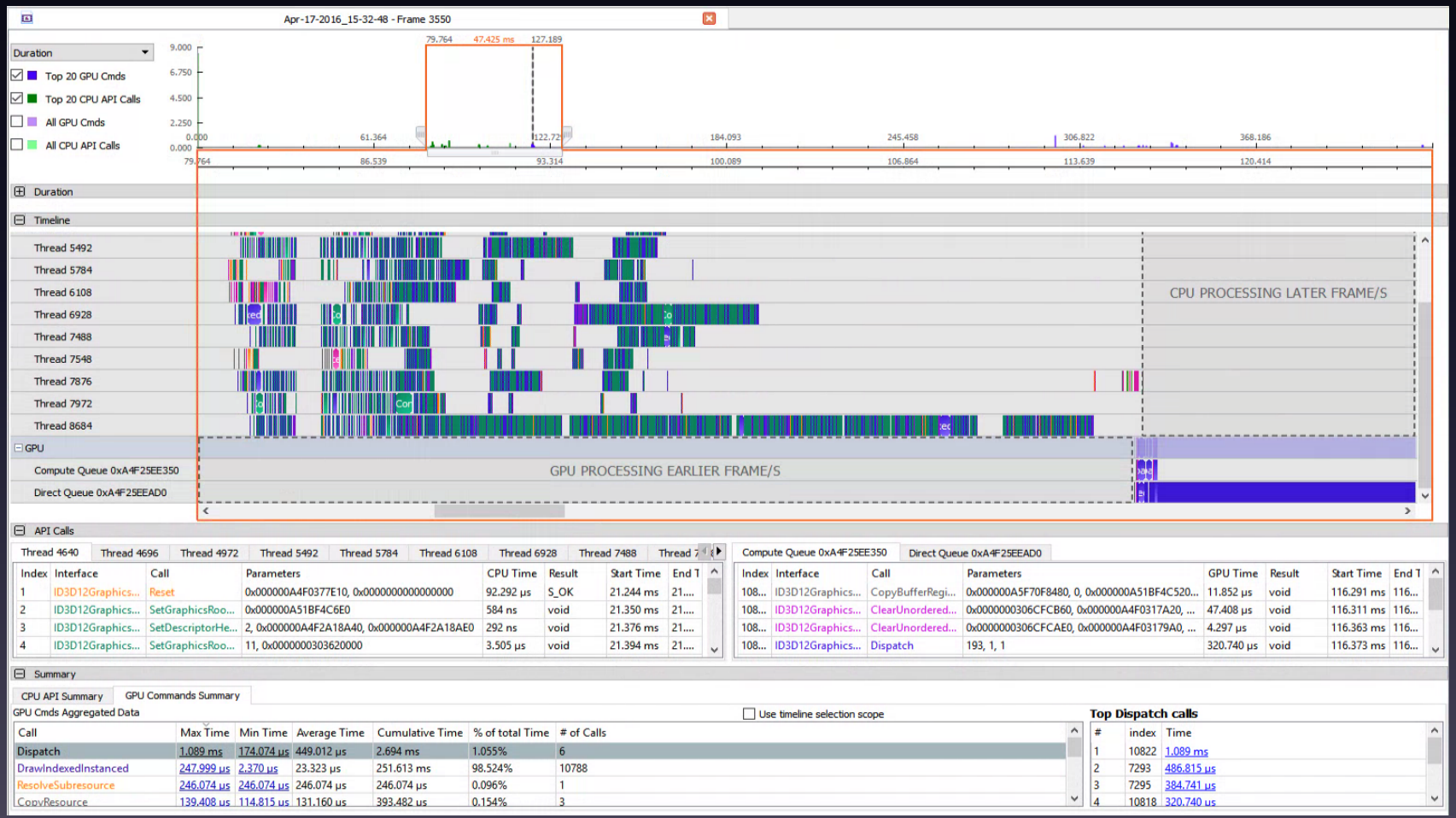
Google Earth



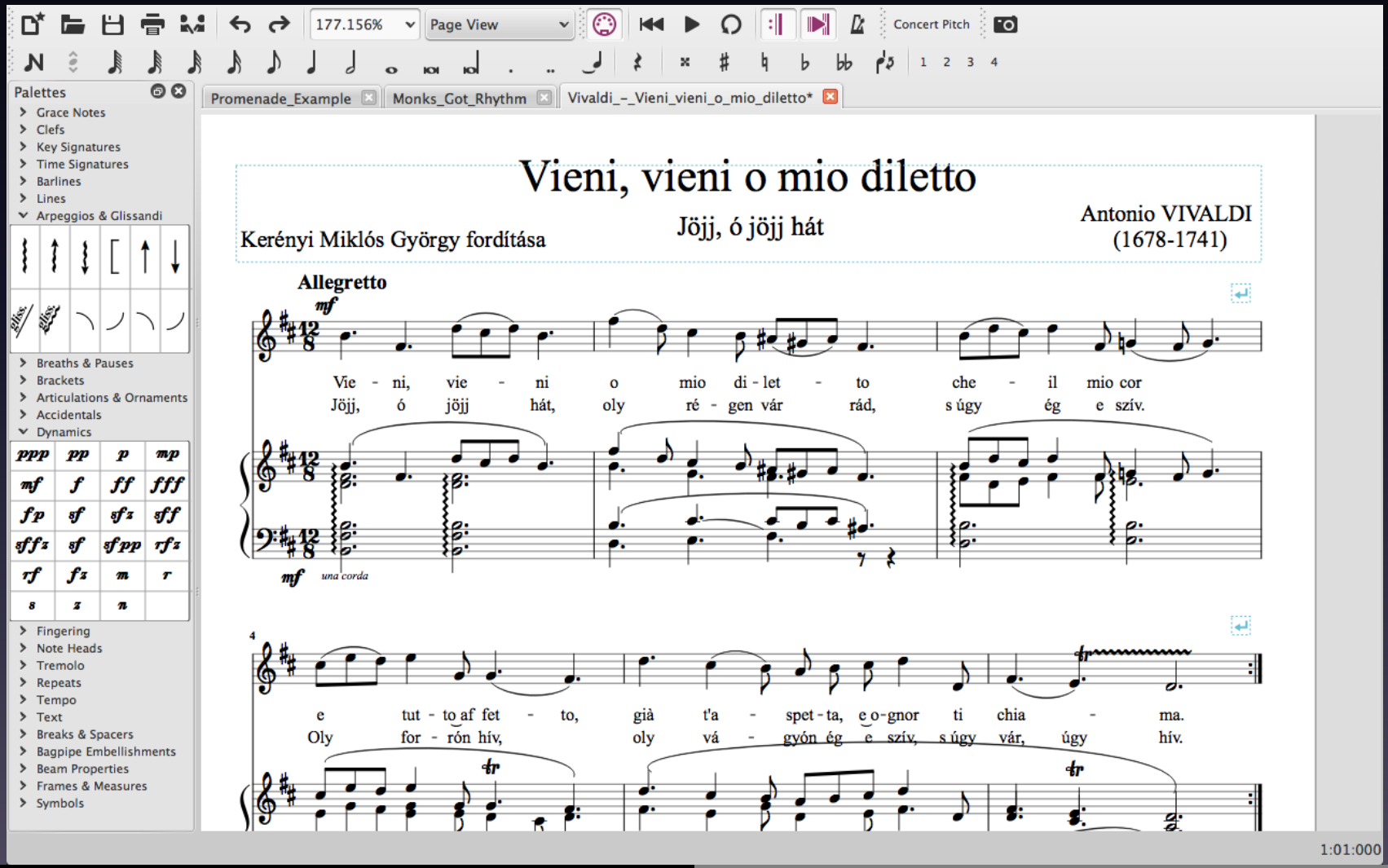
Adobe Photoshop Elements



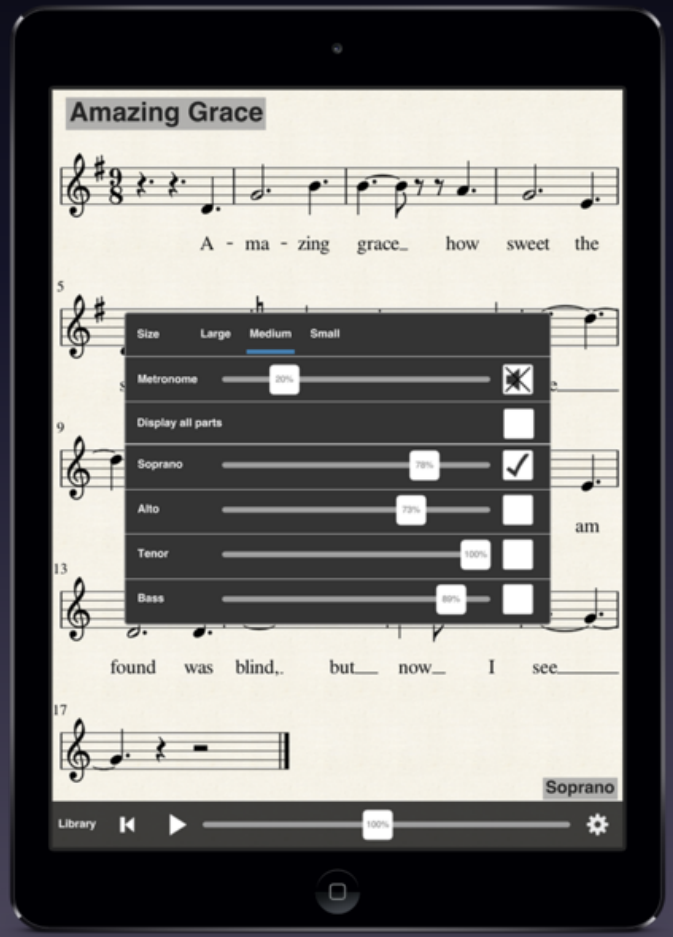
AMD Code XL


Apr-17-2016_15-32-48 - Frame 3550
Duration
Top 20 GPU Cmds
Top 20 CPU API Calls
All GPU Cmds
All CPU API Calls
Timeline
Thread 5492
Thread 5784
Thread 6108
Thread 6928
Thread 7488
Thread 7548
Thread 7876
Thread 7972
Thread 8684
GPU
Compute Queue 0xA4F25EE350
Direct Queue 0xA4F25EEAD0
CPU PROCESSING LATER FRAME/S
GPU PROCESSING EARLIER FRAME/S
API Calls
Summary
CPU API Summary
GPU Commands Summary
GPU Cmds Aggregated Data
Use timeline selection scope
Top Dispatch calls

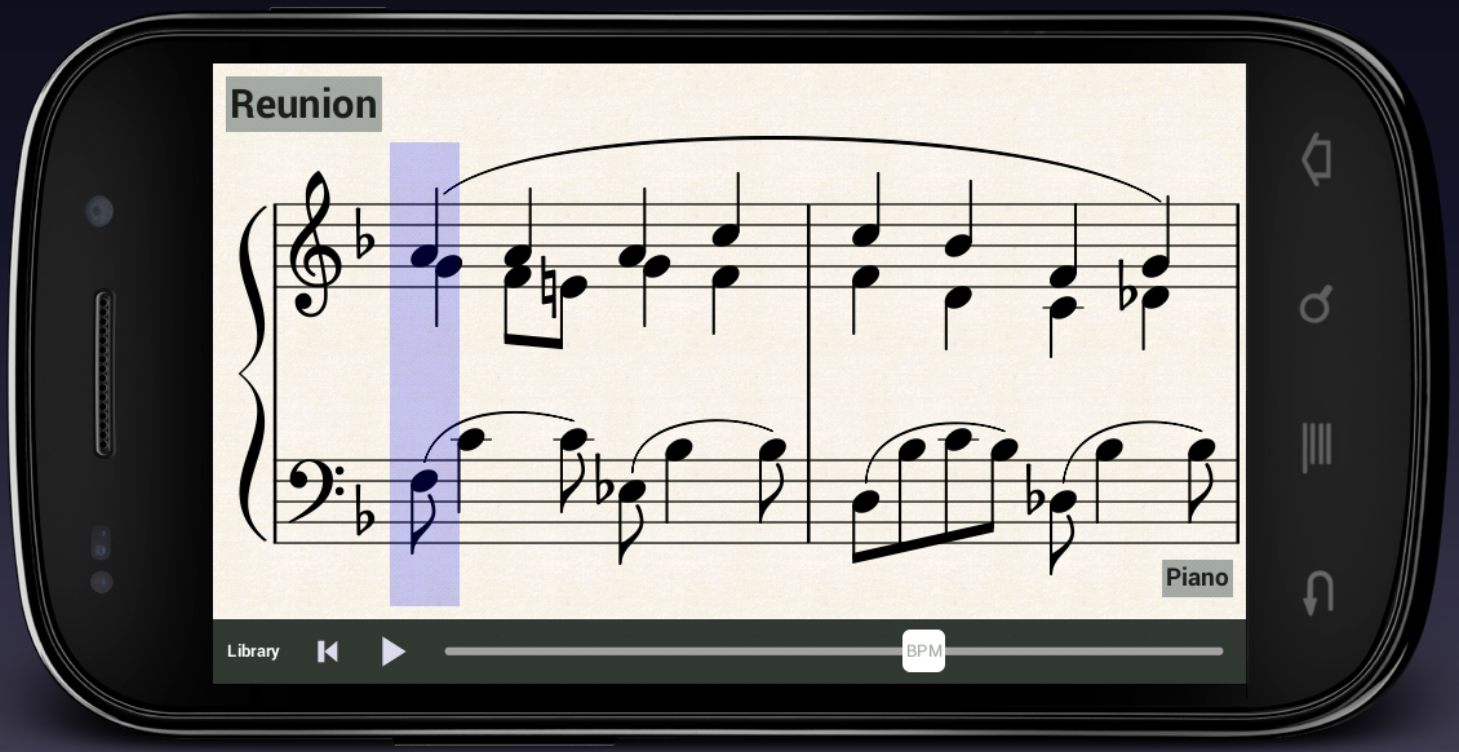
Musescore (Desktop)



Musescore (iOS)



Musescore (Android)



And much more…

Autodesk Maya, DAZ Studio, LightWave 3D, Skype, …

Find out on https://showroom.qt.io/.

And much more…

Autodesk Maya, DAZ Studio, LightWave 3D, Skype, …

Find out on https://showroom.qt.io/.

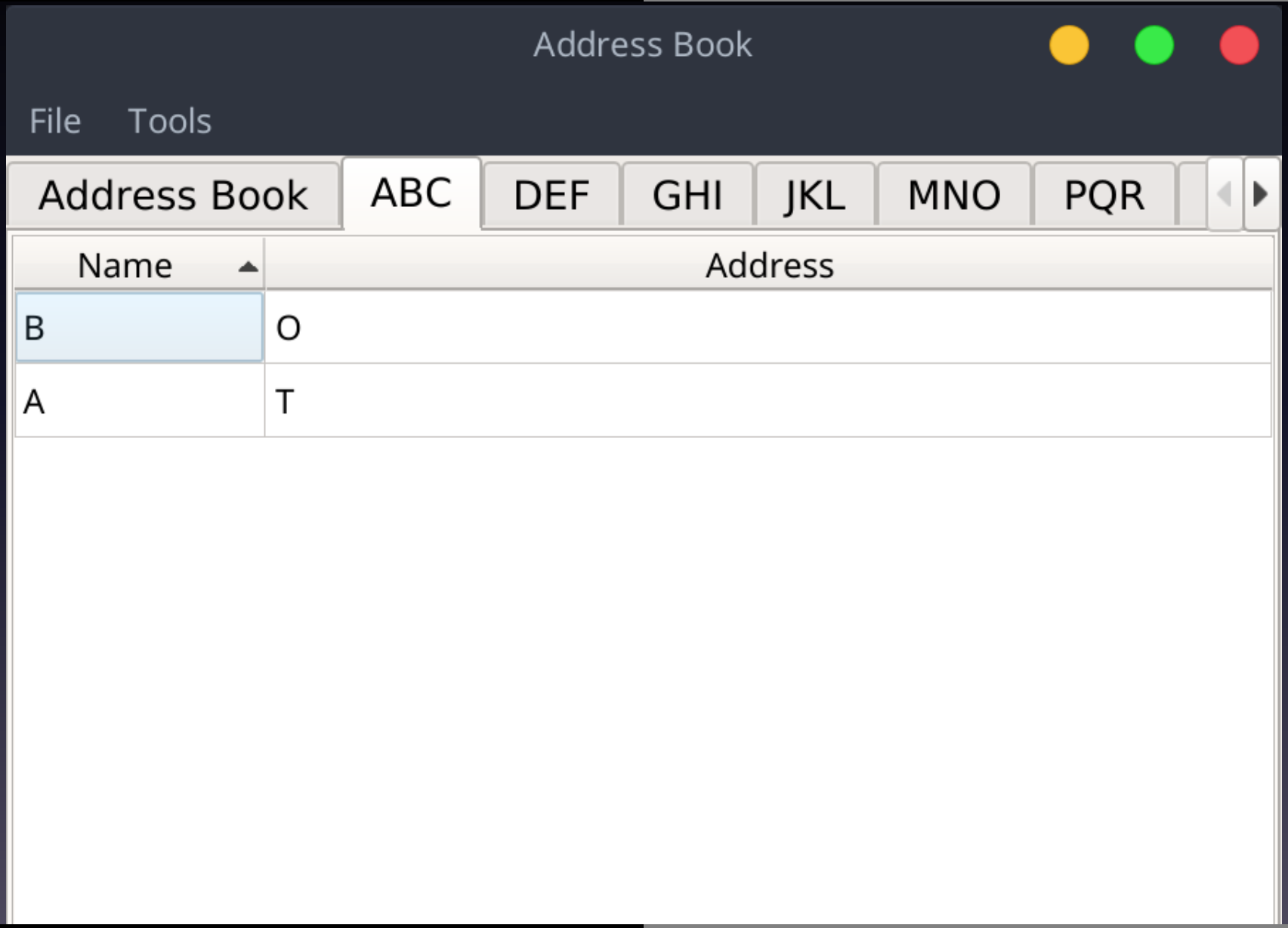
Now, a small demo of OCaml port.

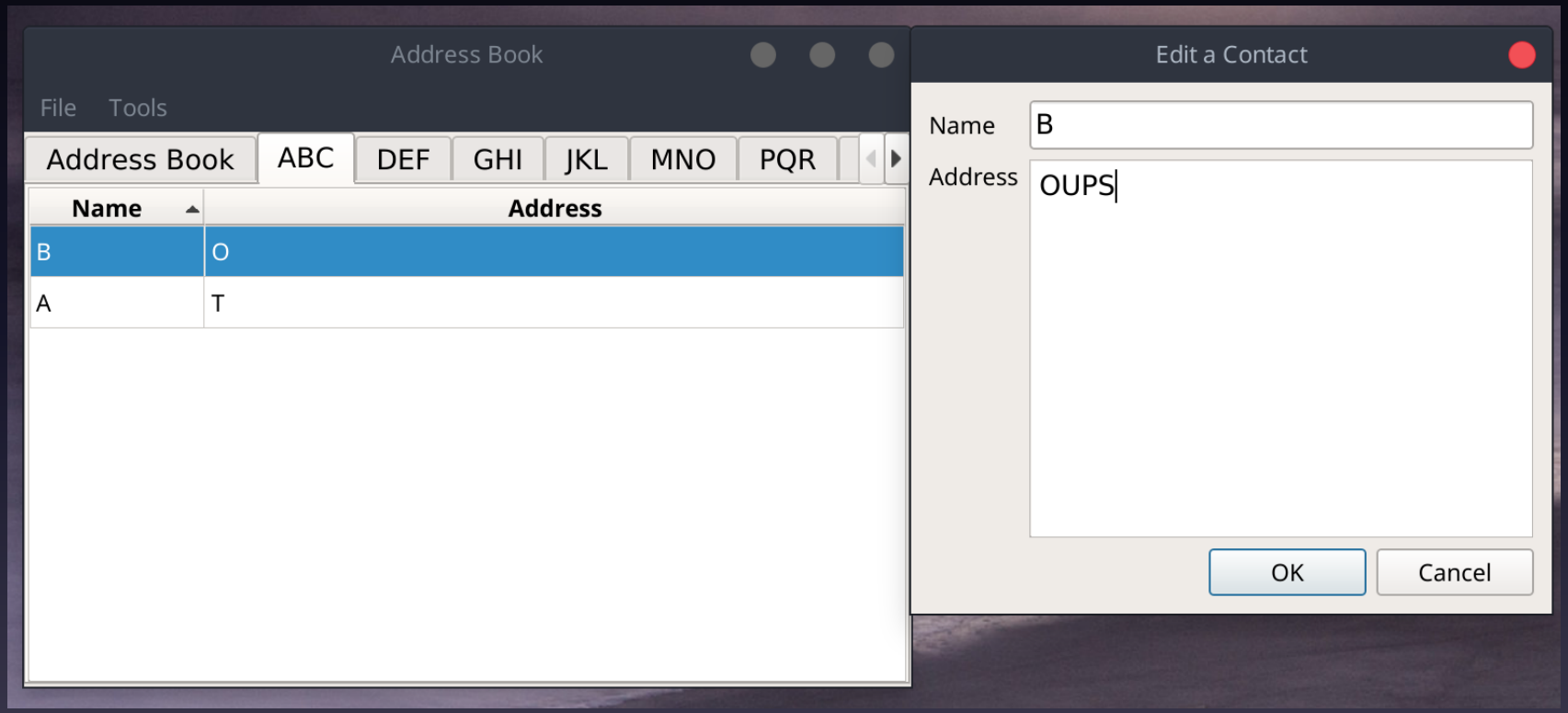
And much more…

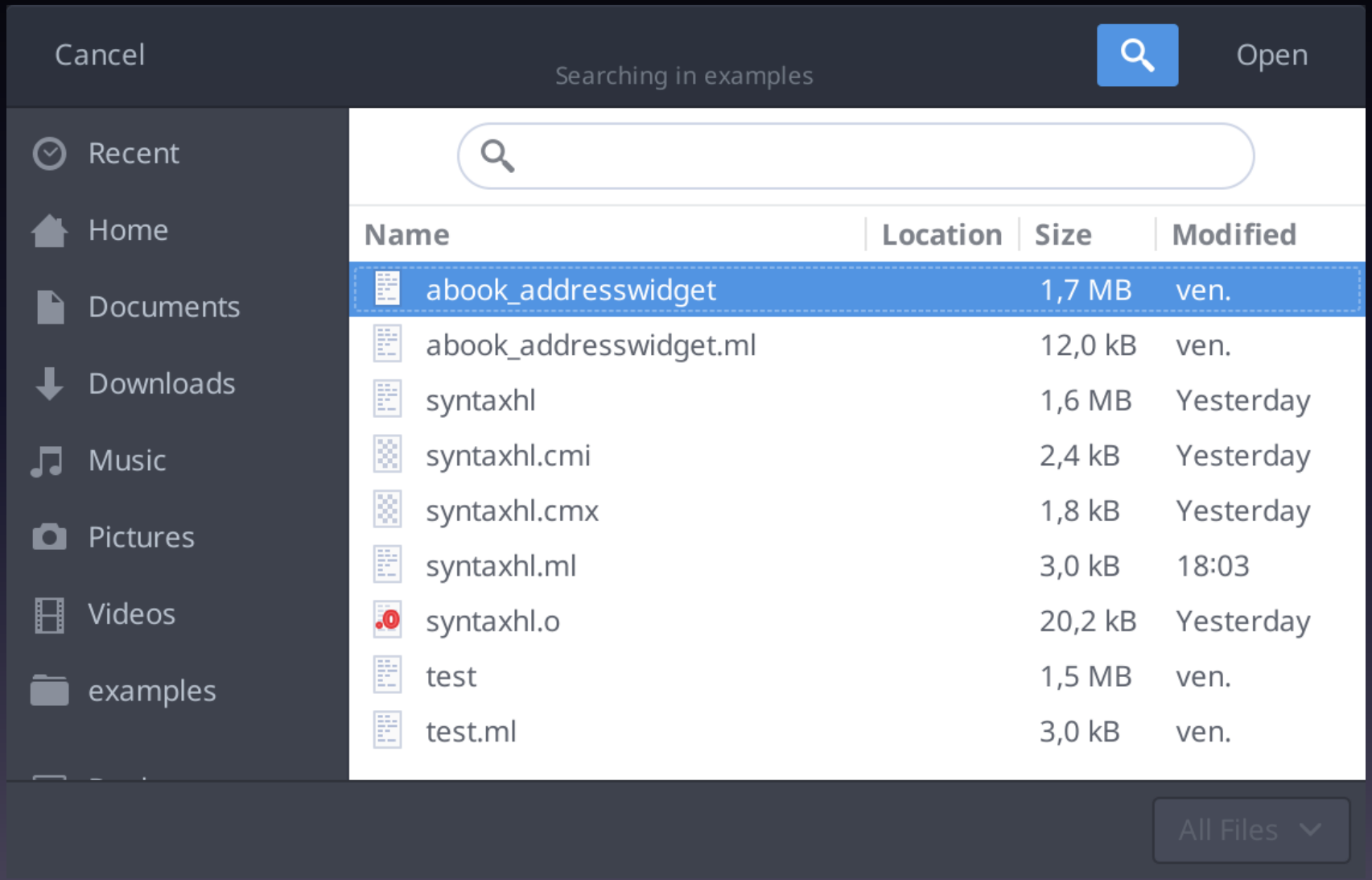
Autodesk Maya, DAZ Studio, LightWave 3D, Skype, …

Find out on https://showroom.qt.io/.

Now, a small demo of OCaml port. Well no.


Address Book
File
Tools
Address Book
ABC
DEF
GHI
JKL
MNO
PQR
Name
Address
B
O
A
T


Address Book
File
Tools
Address Book
ABC
DEF
GHI
JKL
MNO
PQR
Name
Address
B
O
A
T
Edit a Contact
Name
B
Address
OUPS
OK
Cancel


Cancel
Searching in examples
Open
Recent
Home
Documents
Downloads
Music
Pictures
Videos
examples
Name
Location
Size
Modified
abook_addresswidget
1,7 MB
ven.
abook_addresswidget.ml
12,0 kB
ven.
syntaxhl
1,6 MB
Yesterday
syntaxhl.cmi
2,4 kB
Yesterday
syntaxhl.cmx
1,8 kB
Yesterday
syntaxhl.ml
3,0 kB
18:03
syntaxhl.o
20,2 kB
Yesterday
test
1,5 MB
ven.
test.ml
3,0 kB
ven.
All Files

File Help

```
  CHECK_USE_AFTER_FREE(cuite_QModelIndexList_check_use(argself));
  CUITE_Region region;
  value& mlself = cuite_region_register(argself);
  auto self = (QModelIndexList*)cuite_QModelIndexList_from_ocaml(mlself);
  return cuite_int_to_ocaml(self->size());
}

external value cuite_QModelIndexList_isEmpty(value argself)
{
  CHECK_USE_AFTER_FREE(cuite_QModelIndexList_check_use(argself));
  CUITE_Region region;
  value& mlself = cuite_region_register(argself);
  auto self = (QModelIndexList*)cuite_QModelIndexList_from_ocaml(mlself);
  return cuite_bool_to_ocaml(self->isEmpty());
}

external value cuite_QModelIndexList_clear(value argself)
{
  CHECK_USE_AFTER_FREE(cuite_QModelIndexList_check_use(argself));
  CUITE_Region region;
  value& mlself = cuite_region_register(argself);
  auto self = (QModelIndexList*)cuite_QModelIndexList_from_ocaml(mlself);
  self->clear();
  return Val_unit;
}

external value cuite_QModelIndexList_at(value argself,value arg0)
{
  CHECK_USE_AFTER_FREE(cuite_QModelIndexList_check_use(argself));
```

Mapping Qt concepts to OCaml

Qt concepts for data representation:

- plain types (QString, int, float, ...)
- enumerations and flags
- object hierarchies and QObjects
- QVariants

Plain types

- Don't affect memory graph
- Are pure values (physical identity irrelevant)
- Mapped to a concrete OCaml type:

```
void bla(const QString& str);
val bla : string -> unit
```

- Or to an abstract type & a set of functions:

```
val QModelIndex.row : qModelIndex qt -> int
```

Enumeration

Enumeration are directly mapped to a polymorphic variant.

```
enum QStandardPaths::LocationOption {
  LocateFile,
  LocateDirectory
};

type qStandardPaths'LocationOption = [
  | `LocateFile
  | `LocateDirectory
  | `Invalid_value of int
]
```

Flags

Flags are unions of elements from an enumeration. They are manipulated with a generic module and a witness for each flag type.

```
type 'flag set
val define : ('flag -> int64) -> 'flag set

type 'flag t = private int64
val empty : _ t
val set : 'flag set -> 'flag -> 'flag t -> 'flag t
val is_set : 'flag set -> 'flag -> 'flag t -> bool

...
```

Flags

Flags are unions of elements from an enumeration. They are manipulated with a generic module and a witness for each flag type.

```
type 'flag set
val define : ('flag -> int64) -> 'flag set

type 'flag t = private int64
val empty : _ t
val set : 'flag set -> 'flag -> 'flag t -> 'flag t
val is_set : 'flag set -> 'flag -> 'flag t -> bool

...
```

Object hierarchy

Qt has a few **object hierarchies** and restricts it single inheritance the majority of the time.
OCaml encoding preserves the **single inheritance** (the **subtyping** relation is transported).
Multiple inheritance has to be dealt with manually.

Object hierarchy

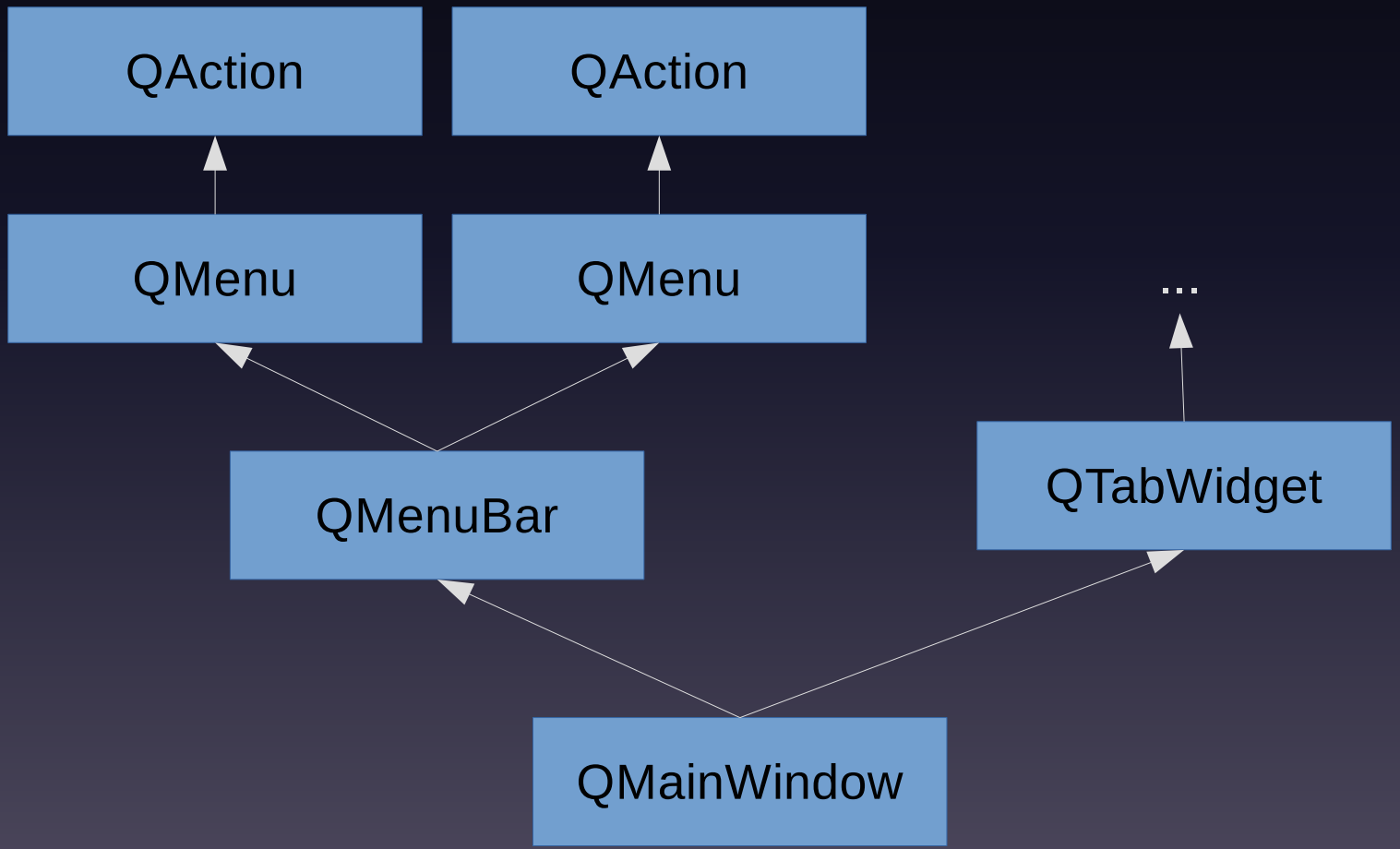
```
(* All qt objects are instances of this scheme *)
type -'a qt

(* QObject is the root of most Qt objects *)
type qObject = [ `QObject ]

(* A QWidget is a QObject *)
type qWidget = [ `QWidget | qObject ]

(* Typechecks ! *)
(some_widget : qWidget qt :> qObject qt)

(* Most functions are polymorphic.
   No coercion needed.
   [show] needs at least a Widget *)
val show : [> qWidget] qt -> unit
```

QObjects

QObject form distinguished hierarchy of objects:

- the offer facilities for automatic and safe **memory management**
- they form a tree of ownership
- they are the only way to introduce **complex heap shapes**
- and **complex control flow**

QObjects


QAction
QAction
QMenu
QMenu
...
QMenuBar
QTabWidget
QMainWindow

QVariant

QVariant is a sum of all plain types : string + int + bool + …
It is mapped to a simple OCaml variant.
Useful to specify data models (more on that later).

Mapping Qt concepts to OCaml

Qt concepts affecting control-flow:

- methods & functions
- signals & slots
- models

Methods & functions

Methods (including static methods) are mapped to simple OCaml functions. They are put in a module named after the class:

```
module QWidget : sig
  ...
  val show : [> qWidget] qt -> unit
  val hide : [> qWidget] qt -> unit
  ...
end
```

Constructors

Constructors are very similar but they are defined in the top module and the naming differs a bit:

```
val new'QPushButton : unit -> qPushButton qt
```

Working around overloading

Qt uses and abuses overloading. Imported functions are disambiguised automatically:

```
val setFormat : string -> ...
val setFormat'1 : QColor -> ...
val setFormat'2 : QFont -> ...
```

A **curation** work is needed, to remove some variant or pick better names.

Signals and slots

The main primitive for dynamic control flow:

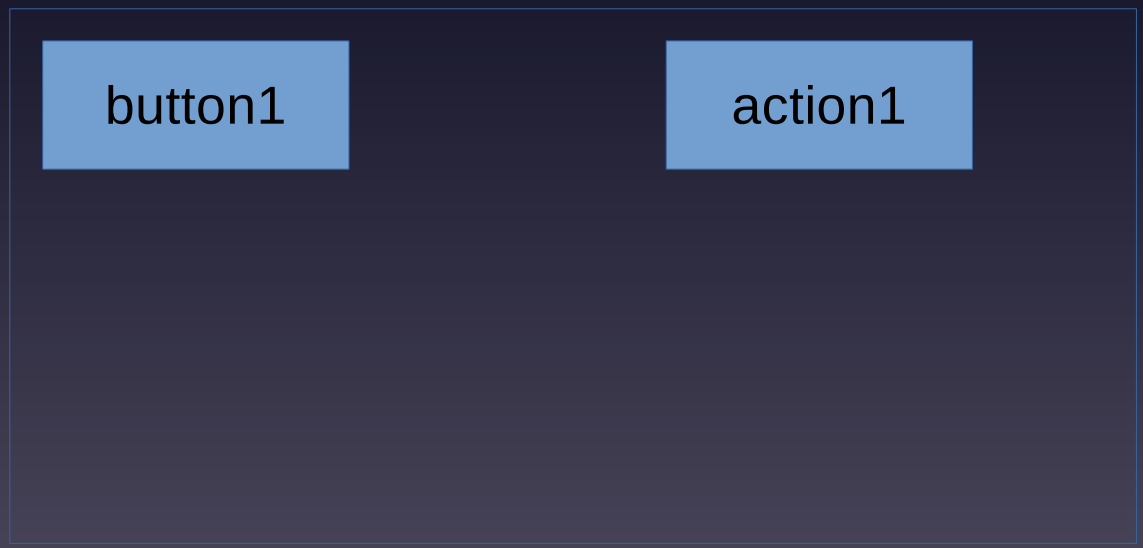
- a descendant of QObject can define signals and slots
- a **signal** can be connected to **slots** and **closures**
- a signal can be emitted
- the slots and closures that are connected are executed

Signals and slots

```
type (-'a, +'b) signal
type (-'a, -'b) slot

val connect_slot :  'a t -> ('a, 't) signal
                 -> 'b t -> ('b, 't) slot -> unit

val connect :  'a t -> ('a, 't) signal
            -> ('t -> unit) -> unit
```

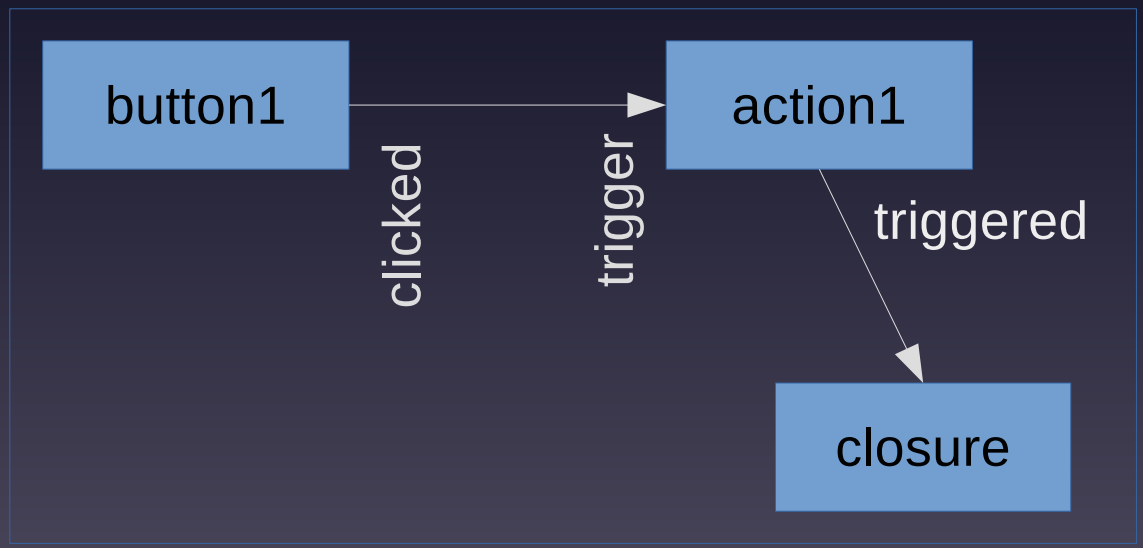
Signals and slots

```
connect_slot button1 (QButton.signal'clicked())
             action1 (QAction.slot'trigger());

connect action1 (QAction.slot'triggered())
  (fun _ -> print_endline "Action!");
```



Signals and slots

```
connect_slot button1 (QButton.signal'clicked())
             action1 (QAction.slot'trigger());

connect action1 (QAction.slot'triggered())
  (fun _ -> print_endline "Action!");
```



Models

Models are used to specify dynamic **data source**.

An abstract model is a QObject with one or more virtual methods that define **observations on a set of data**:

- how many items
- content or display properties of each item
- structure of the items: tree-like, tabular, etc

Models

Each Model has to be encoded manually in OCaml.
The encoding is in two parts:

- the class is inherited only once on C++ side
- an instance wraps an OCaml record that contains a closure for each method

No C++ has to be written by the user :).

Models

Example:

- QOCamlTableModel
- QOCamlSyntaxHighlighter

Models

```
type 'a qOCamlSyntaxHighlighter =
  [ `QOCamlSyntaxHighlighter of 'a | qSyntaxHighlighter ]

type 'a qOCamlSyntaxHighlighter'callback = {
  highlight_block : 'a -> 'a qOCamlSyntaxHighlighter qt -> string -> unit;
}

val new'QOCamlSyntaxHighlighter :
  'a qOCamlSyntaxHighlighter'callback -> 'a -> 'a qOCamlSyntaxHighlighter qt

module OCamlSyntaxHighlighter : sig
  val setFormatColor : 'a qOCamlSyntaxHighlighter qt ->
                       int -> int -> [> qColor] qt -> unit =
end
```

Managing QObject

Four cases for memory management:

- **Relocatable objects** (plain types not mapped to a native OCaml type). Safe, automatic, don't affect graph.

Managing QObject

Four cases for memory management:

- **Relocatable objects** (plain types not mapped to a native OCaml type). Safe, automatic, don't affect graph.
- **QObject**. Safe, automatic, one region released at a time.

Managing QObject

Four cases for memory management:

- **Relocatable objects** (plain types not mapped to a native OCaml type). Safe, automatic, don't affect graph.
- **QObject**. Safe, automatic, one region released at a time.
- **Scarce resources** (e.g sockets, GPU buffers…). Safe, manual management but GC can catch mistakes.

Managing QObject

Four cases for memory management:

- **Relocatable objects** (plain types not mapped to a native OCaml type). Safe, automatic, don't affect graph.
- **QObject**. Safe, automatic, one region released at a time.
- **Scarce resources** (e.g sockets, GPU buffers…). Safe, manual management but GC can catch mistakes.
- **Raw pointers** (e.g. iterators). Unsafe, manually managed, should ideally be hidden behind an OCaml-ish abstraction

Managing QObject

Four cases for memory management:

- **Relocatable objects** (plain types not mapped to a native OCaml type). Safe, automatic, don't affect graph.
- **QObject**. Safe, automatic, one region released at a time.
- **Scarce resources** (e.g sockets, GPU buffers…). Safe, manual management but GC can catch mistakes.
- **Raw pointers** (e.g. iterators). Unsafe, manually managed, should ideally be hidden behind an OCaml-ish abstraction

Convenience/performance trade-off: when possible, release manually.

Managing QObject

One primitive for all objects :

```
val delete : _ qt -> unit
```

- release Object memory
- for QObject, also releases all children
- detect use after free
- but cannot track aliasing of raw pointers :-(

Getting started

Install qt5 and cuite package.

```
# yaourt -S qt5
# opam pin add cuite
    https://github.com/let-def/cuite.git
```

Getting started

Install qt5 and cuite package.

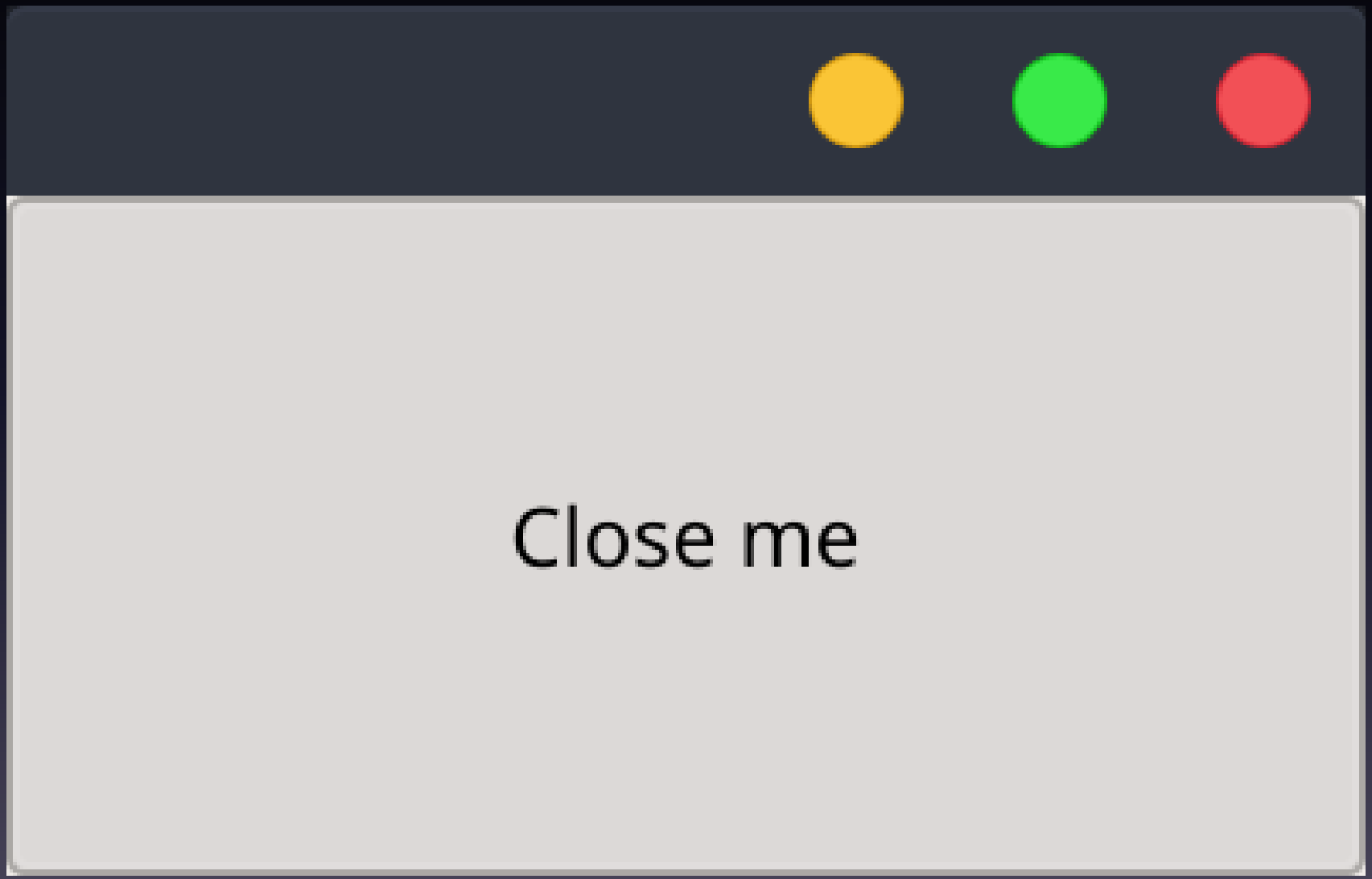
```
# yaourt -S qt5
# opam pin add cuite
    https://github.com/let-def/cuite.git
```

Portability not a concern yet:

- tested only with **Qt 5.9**
- rely on **pkg-config** for finding Qt5
- rely on **g++** for compilation
- rely on **-rpath** linker option for finding dynamic libraries

Minimal example

```
open Cuite

let () =
  let app = new'QApplication Sys.argv in
  let window = new'QMainWindow None QFlags.empty in
  let button = new'QPushButton'1 "Close me" None in
  QMainWindow.setCentralWidget window button;
  Qt.connect_slot'
    button (QPushButton.signal'clicked1())
    app (Qt.slot_ignore(QApplication.slot'quit()));
  QWidget.show window;
  exit (QApplication.exec ())

# ocamlfind opt -linkpkg -package cuite -o test test.ml
```


Close me

Approach

- Manually map primitive concepts (objects, classes, methods, signals, slots, variants, …)

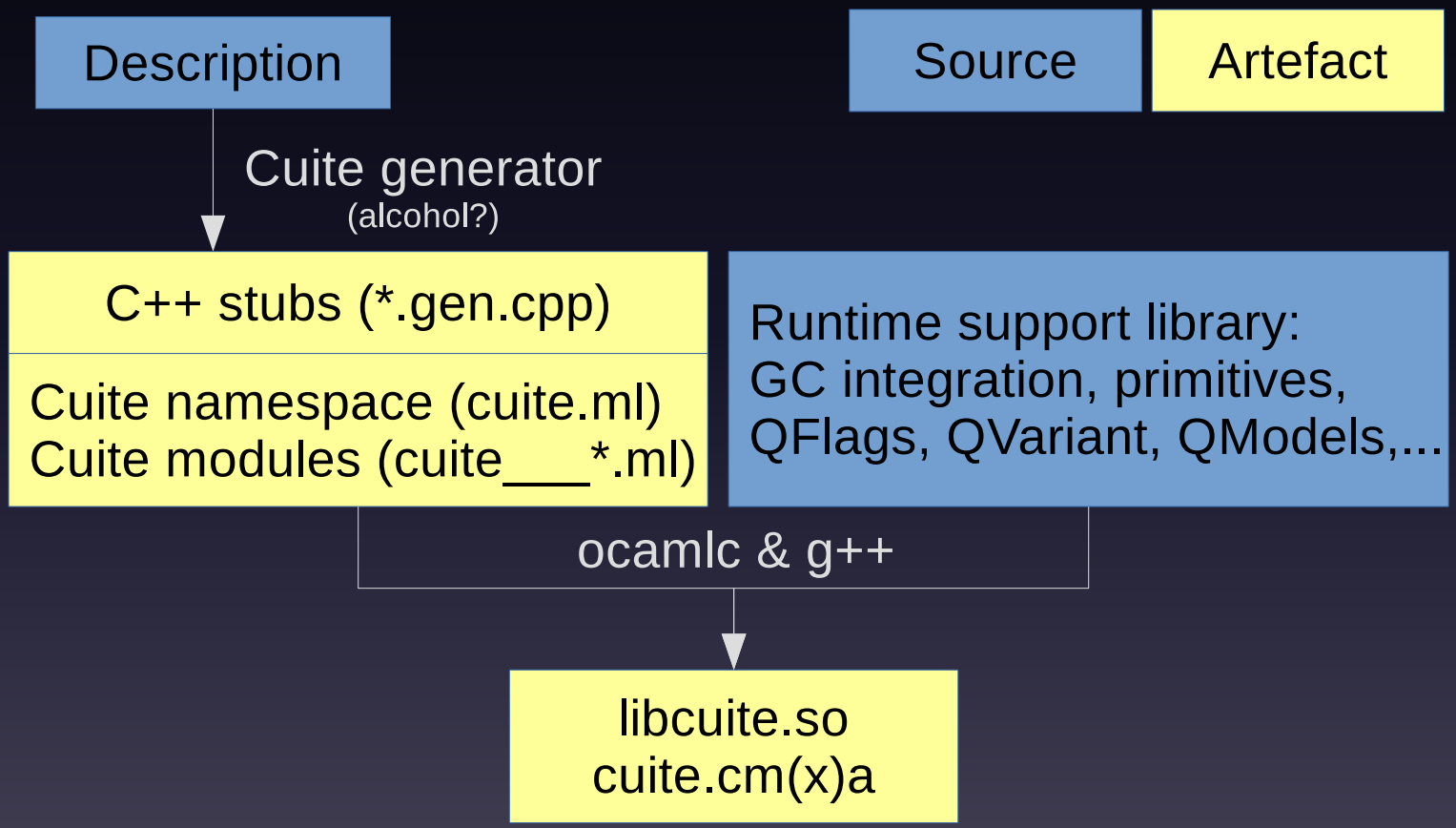
Approach

- Manually map primitive concepts (objects, classes, methods, signals, slots, variants, …)
- Automatically generate compositions of these concepts (taking an abstract description of Qt has OCaml values)

Approach

- Manually map primitive concepts (objects, classes, methods, signals, slots, variants, ...)
- Automatically generate compositions of these concepts (taking an abstract description of Qt has OCaml values)
- Manually handle corner cases (e.g models)

Automatic generation



Future work

A "proof-of-concept" has been made for each of these concepts.

Future work

A "proof-of-concept" has been made for each of these concepts.
The majority of Qt Gui/Widgets library is available…but a few crucial parts are still missing.

Future work

A "proof-of-concept" has been made for each of these concepts.
The majority of Qt Gui/Widgets library is available…but a few crucial parts are still missing.
Never ending work: too much to be handled alone, better to fix/extend as new use cases present themselves.

Future work

Things to investigate:

- Qt-designer support (seems "easy", a PPX could do)
- Lwt/Async integration
- targetting mobile platforms
- a multi-threading story?
- cleaning up ad-hoc polymorphism/overloading mess

Conclusion

Thanks for your attention!

Don't hesitate to ask for guidance if you are interested in making use of the lib or contributing.

(-: