Fast QML UI prototyping for platforms WITHOUT Qt/QtQuick support

Attila Csipa
@achipa
Qt DevDays
Berlin, 08.10.2013

Session content

- What (Fast UI Prototyping without QtQuick)
- Why do it?
- Why NOT do it?
- How to do it?
- Case study – Nokia Asha
- Case study – Console UI

Fast UI development

- Terse, but readable syntax
- Declarative UI
- Quick iteration cycles

QtQuick + QML = Declarative + terse syntax + quick iteration

However, not all platforms have (good) Qt(Quick) support

I want QtQuick, but...

Qt Developer Days 2013

Policy

- No interpreted/dynamic code
- No V8/JS engine
- Because someone said “NO”

The swimsuit police checking the length of a suit, 1922

I want QtQuick, but...

No hardware accelerated OpenGL (ES) capability (or drivers)

- A hard requirement for QtQuick 2.x

A bicycle with 12 rockets mounted on the back wheel. ~1930s

I want QtQuick, but...

- Resource limitations
  - Not enough memory
  - Not enough bandwidth
  - Not enough disk space
  - Not enough…

Cheese “sandwich” served on the Sao Paolo – Manaus TAM flight

I want QtQuick, but…

- No native look and feel
- No native QtQuick-based component set
- No Integration with target platform

So why do it?

- Business angle
- Technology angle
- Development angle

Why do it?
Business angle

- Gazillion of non-Qt(Quick) capable devices
- A lot of those without UI prototyping tools
- Foot in the door! (follow-up projects)
- Accessibility (where not yet covered by Qt)
- (For fame and glory)

Why do it?
Tech angle

- Minimize resource usage in case of remote UIs
- Be able to chose UI tech best suited for user interaction
- State of Qt port on target platform not a showstopper
- Multi-UI applications
  - Usable both locally with a GUI, and remotely (via telnet/SSH)

Why do it?
Development angle

- Reuse knowledge of QML
- Complement lacking IDEs with QtCreator
  - Syntax highlighting, autocomplete, help, potentially debugging
- Single language interface between developer & designer
- Less people need to know platform specifics
- Faster (less painful?) design iterations

Helmet test, ca 1912

Why not do it?

- Maintenance burden
- Hard to upstream
- Flexibility
- There is a reason QtQuick(2) exists, we are talking primarily about PROTOTYPING
- “World doesn’t need another piece of crap”
  Dan Dodge, Qt DevDays QNX Keynote

The last four couples standing in a dance marathon. Chicago, c. 1930

The approaches – analysis

- Workflow
- Prerequisites
- Advantages
- Disadvantages
- Problem Domain

Example approaches

- Roll your own (QtQuick)
- Client side QML
- Code Generation
- [your idea here – the previous ones are just examples!]

Think out of the box!

Approach #1 Roll your own

QML Types Provided By The QtQml Module

The QtQml module provides the definition and implementation of various convenience types which can be used with the QML language, including some elementary QML types which can provide the basis for further extensions to the QML language.

The QtQml module provides the QtObject and Component object types which may be used in QML documents. These types are non-visual and provide building-blocks for extensions to QML.

Workflow – simple, exactly the same as with QtCreator and QtQuick!

Approach #1 Roll your own

- Prerequisites
  - Qt on target platform, with functional QtQml
- Advantages:
  - Leverage JavaScript and bindings via Qt
  - Easy event handling (signals/slots)
  - QML debugging from QtCreator
- Disadvantages:
  - Requires Qt and QtQml on target platform

Roll your own
Applicable problem domain

- Suitable for simple problem domains
  - Text/console mode
    - CDK/ncurses interface
  - Custom hardware (LED magic!)
    - Beagleboard, blinkenlights

Projekt Blinkenlights, Berlin, 2001 - view from Berliner Fernsehturm
Photo by Tim Pritlove

Approach #2 Client side QML

- Create QML in QtCreator
- Run
  - Strip import statements and any JS
  - Deploy resulting QML file sync with device
  - Via File system or
  - Via Network protocol/socket
- Application on device (re)loads QML and constructs UI
  - Feels almost like live-editing!
  - If you do want live-editing, you will need to save state/values!
- Rinse and repeat

Approach #2 Client side QML

- Prerequisites
  - Shared data channel to client (network, storage…)
  - Implemented (or wrapped) component toolkit
- Advantages
  - Does not require Qt on target platform at all!
- Disadvantages
  - Only for really basic UIs
  - Lot of work (as no code reuse can happen)
  - No JavaScript or bindings
  - Difficult to debug
  - Very good understanding of target platform required

Client side QML
Applicable problem domain

- Platforms with no Qt support
- Static UI design (no JS!)
- Mockups

Client side QML

Example: Java ME with Tantalum


```
public void constructUI(final byte[] JSONdata) {
  JSONObject o;
  try {
    o = new JSONObject(new String(JSONdata));
  } catch (JSONException ex) {
    L.e("bytes are not a JSON object", "featURL", ex);
    return null;
  }

  try {
    final JSONObject feed = ((JSONObject) o).getJSONObject("ApplicationWindow");
    entries = o.getJSONArray("Options");
    for (int i = 0; i < entries.length(); i++) {
        final JSONObject m = entries.getJSONObject(i);
        final String OptionLabel = m.getJSONObject("Option").get("text");
        displayable.addCommands(new Command(OptionLabel, Command.ITEM, 1));
    }
    if (entries.length() > 1)
        displayable.addCommandListener(this);

  } catch (JSONException e) {
    L.e("JSON no ApplicationWindow", "featURL", e);
  }
}
```

Approach #3 Code generation

- Create QML in QtCreator
- Run
  - Component output constructs source code based on QML
  - ApplicationWindow (or QtCreator platform plugin) compiles code
  - Packaging
  - Deploy to device/simulator
  - Execute on device (if possible)
  - Live-edit-like development possible, like in previous case (if code can be loaded dynamically on target platform)
- Rinse and repeat

Approach #3 Code generation

- Prerequisites
  - Qt and target platform *TOOLS* running on same device
- Advantages
  - Customizability
- Disadvantages
  - JavaScript and a suitable binding availability not guaranteed
  - Complexity
  - Maintenance burden

Code generation
Applicable problem domain

- Platforms with no Qt support at all
- Light logic can be included, client platform permitting
  - Simple bindings can be simulated
  - JavaScript may or may not be present

Code generation is in effect…

… source-code level (de)serialization!

Case Study #1

Nokia Asha Software Platform

Series40 (which is NOT Symbian)

Qt Developer Days 2013

First device in 1999,
the Nokia 7110

(but don't worry,
Qt is actually
4 years older ;)

A Coca Cola company delivery truck in Knoxville, 1909.

A few years later…

Qt Developer Days 2013

1.5 billion devices by January 2012

650 million active (plenty of even touch devices)

Freemium and ads DO work

North London Derby between Arsenal and Tottenham Hotspur at Highbury, 1934

But the world changed

Women on motorcycles in Great Britain, 1930s

New Nokia Asha

Qt Developer Days 2013

=

Series40 Hardware adaptation

+

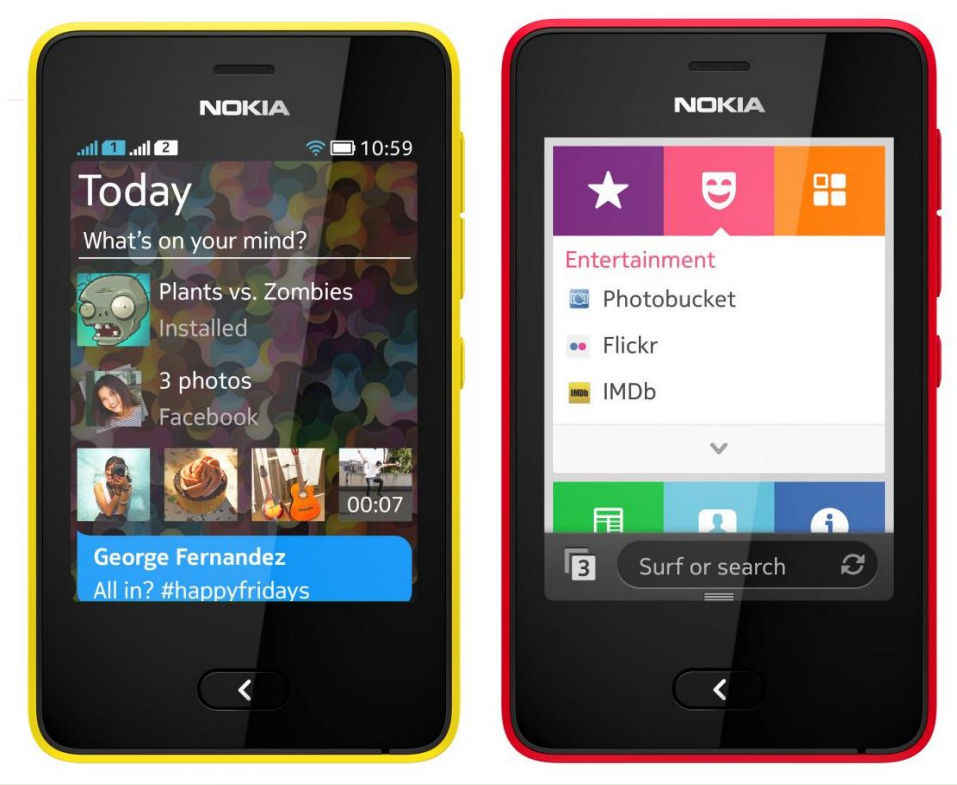
Smarterphone middleware

+

Swipe UI

Nokia Asha Developer Offering

- Nokia Asha SDK 1.0 (Java ME)
  - Java ME MIDP 2.1, CLDC 1.1
  - Optional JSRs
  - Nokia APIs
  - Max JAR file size: 5 Mb
  - Max Java Heap: 3 Mb
- Nokia Asha web app tools 3.0.0
- Xpress Web App Builder 1.0



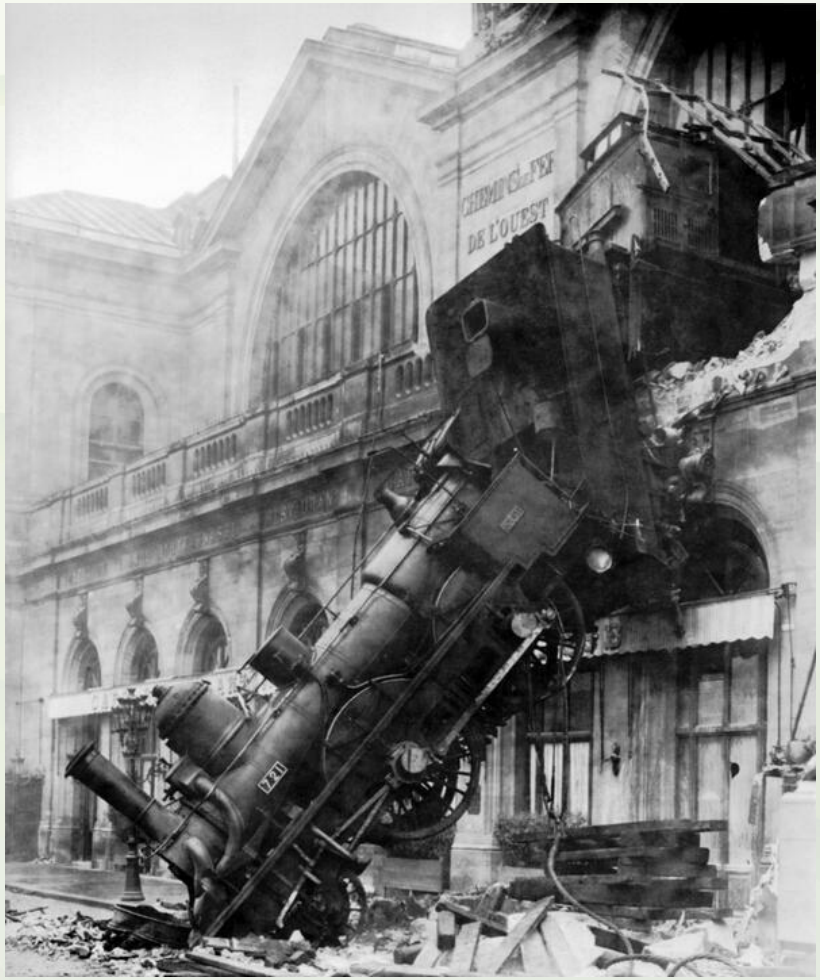
Feeling resource constrained yet?

Under 8 megs of application RAM, no native code, no OpenGL

What is Qt doing in this story?

Qt Developer Days 2013

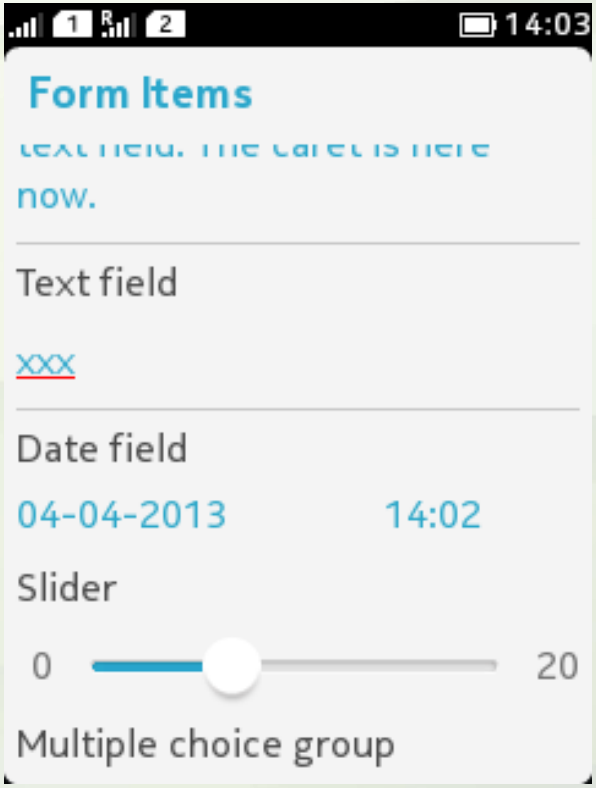
…let's take a closer look before we jump to conclusions



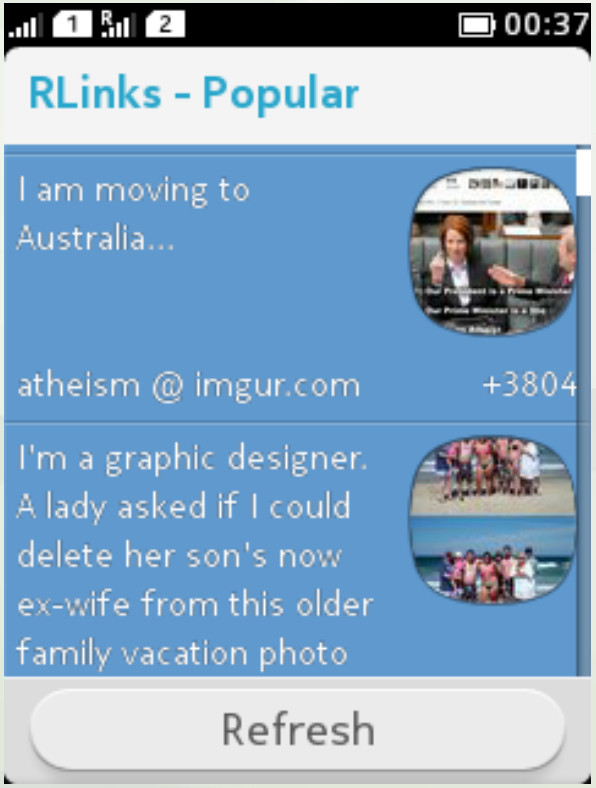
Train wreck at Montparnasse Station. Paris, 1895.

Understanding Java ME UI

Canvas



LCDUI



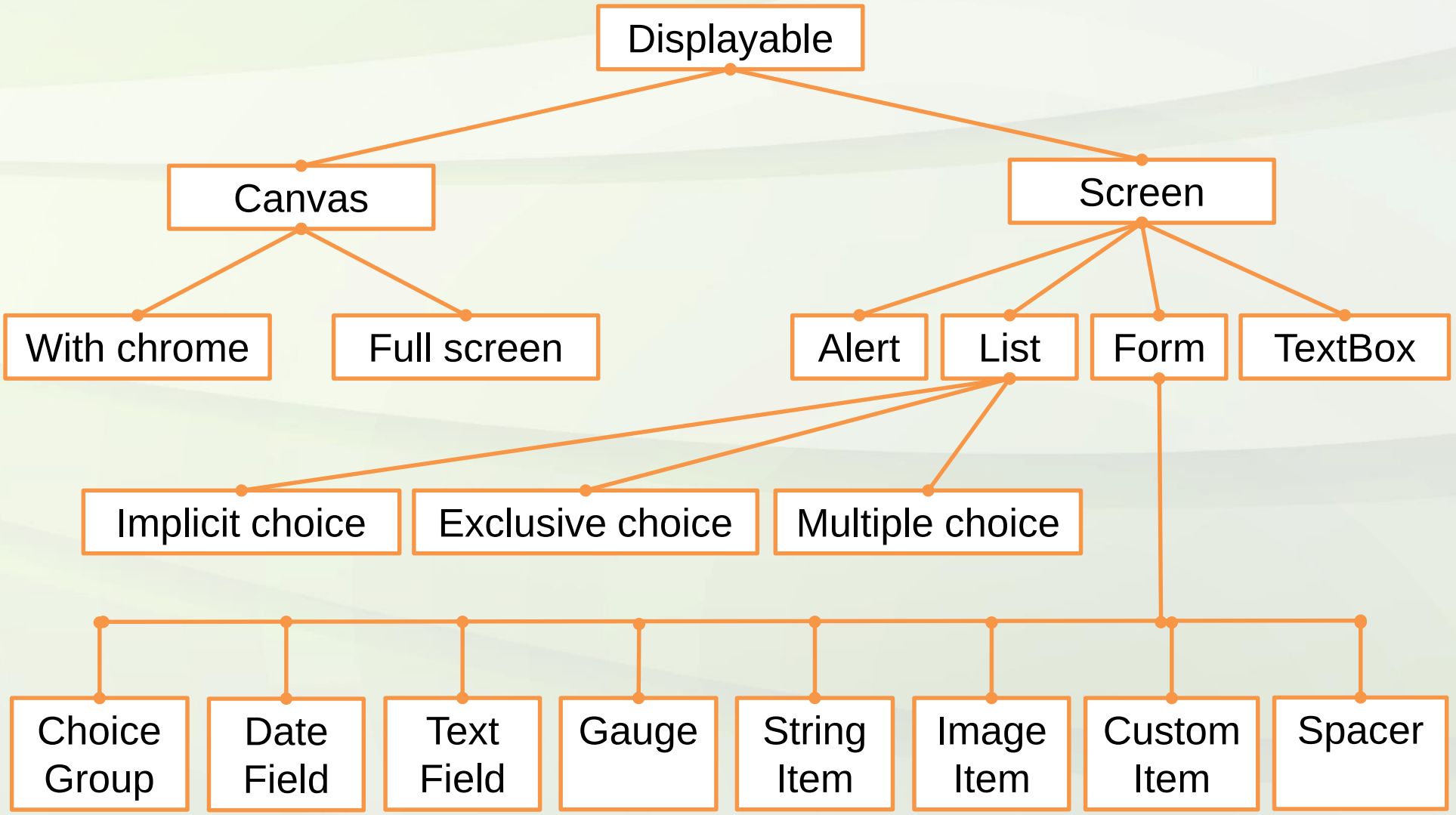
LWUIT

The key: native look and feel

LCDUI

- High-level components
- Nokia UI API
- Asha look & feel
- No customizability
- (except CustomItem)

Simplicity can be an advantage



Which approach to use?

Case study #1 Nokia Asha

- Custom components
  - No Qt/QtQml
  - No native look and feel
  - Too large memory footprint
  - Slow JavaScript performance
- Code generation
  - Java ME has no reflection (or classloaders)
  - Still, possible with application reloads
- Client side QML
  - JSON parser exists (f.ex as part of Tantalum)
  - Native look and feel, even fairly simple with LCDUI

LCDUI (Java ME) vs QML


```
…
  Form form = new Form(“Hello world”);
  Image image = null;
  try {
    image = Image.createImage(file);
  }
  catch (NullPointerException npe) {
    System.out.println("No file name specified");
  }
  catch (IOException ioe) {
    System.out.println("Image not found: " + file);
  }
  form.append(“First!”);
  form.append(image);
  form.addCommand(Commands.BACK);
  form.setCommandListener(this);
  Display.getDisplay(this).setCurrent(list);
…
```

3 classes… 6 methods… 260 lines of code…

```
import com.nokia.asha.lcdui 1.0

ApplicationWindow {
  Form {
    header: “Hello World!”
    StringItem { text: “First!” }
    }
    Image {  src: “hello.png”  }
    Options: [
       Option {
         text: “Back”
         type: BACK
       }
    ]
  }
}
```

…That’s all!

The first successful run

Annie Edison Taylor
The first person to survive going over Niagara Falls in a barrel, in 1901

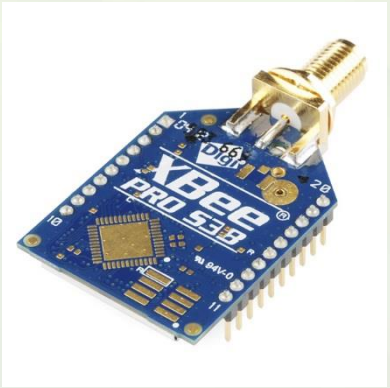
Case study #2
Embedded Remote Sensing

Raspberry PI

Qt-enabled Linux distros available

ARM11 + OpenGL ES

+



X-Bee Radio module

Superior LOS range – up to 48km

9600 bps data rate

CONNECT 9600

- Command line interfaces – Console UIs
  - Interfaces based on [n|pd]curses or Newt, CDK, NDK++
  - Pretty old, none declarative – scripted at best (dialog)
  - …but still useful…

Also a bit resource constrained

- Low resource usage
  - Bandwidth
    (ideal for SSH)
  - Memory
  - Distributable size

Cheese “sandwich” served on the Sao Paolo – Manaus TAM flight

Let’s pick a toolkit – CDK

- Short for Curses Development Kit

http://www.tldp.org/HOWTO/NCURSES-Programming-HOWTO/tools.html

19.1.3. Conclusion

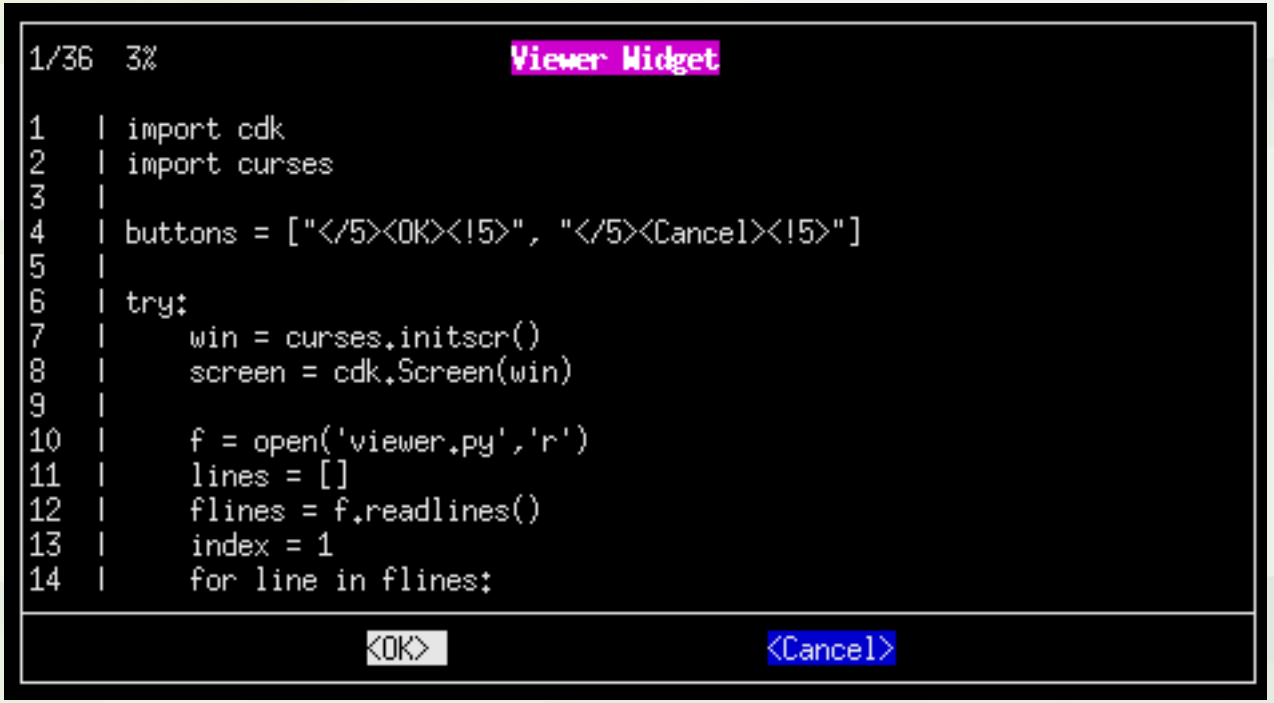
All in all, CDK is a well-written package of widgets, which if used properly can form a strong frame work for developing complex GUI.

Console UI – CDK

21 curses-rendered (text mode) widgets

Alphalist	Button	Buttonbox	Calendar	Dialog
Entry Field	File Viewer	File Selector	Scale	Slider
Graph	Histogram	Item List	Label	Matrix
	Marquee	Pulldown Menu	Template	

Console UI – CDK



Which approach to use?

Case study #2 Embedded remote sensing solution

- Custom components
  - Qt/QtQml present
  - Widgets present (CDK)
  - Simple enough UI for memory/JS considerations
  - Platform does not have a “native look and feel” = our choice
- Code generation
  - Large number of configurable widgets = complexity
  - No JavaScript
- Client side QML
  - Large number of configurable widgets
  - More effort than custom components
  - JSON parser exists

CDK (native) vs QML


```
CDKSCREEN *cdkscreen;
CDKLABEL *demo;
WINDOW *cursesWin;
const char *mesg[4];

cursesWin = initscr ();
cdkscreen = initCDKScreen (cursesWin);
initCDKColor ();
mesg[0] = "</5><#UL><#HL(30)><#UR>";
mesg[1] = "</5><#VL(10)>Hello World!<#VL(10)>";
mesg[2] = "</5><#LL><#HL(30)><#LR>";

demo = newCDKLabel (cdkscreen,
  CDKparamValue (&params, 'X', CENTER),
  CDKparamValue (&params, 'Y', CENTER),
  (CDK_CSTRING2) mesg, 3,
  CDKparamValue (&params, 'N', TRUE),
  CDKparamValue (&params, 'S', TRUE));

setCDKLabelBackgroundAttrib (demo, COLOR_PAIR (2));
```

```
import org.cdk.widgets 1.0
ApplicationWindow {
  Label {
    anchors {
      horizontalcenter: parent.horizontalcenter
      verticalcenter: parent.verticalcenter }
    width:  30
    height: 10
    text:   “Hello World!”
    border: true
    bordercolor: 5
    color:  2
  }
}
```

Potential targets

- The Web
- Android (via declarative XML)
- Windows 8 (via XAML)
- [Favorite hardcore platform here]

Men shaving, ~1940s

Takeaway

It’s not about what platform Qt supports…

…It’s about where you can take Qt with you

Thank you!

Qt Developer Days 2013

Questions?

Attila Csipa
@achipa

Fast QML UI prototyping for platforms
WITHOUT Qt/QtQuick support