An Application Manager on a GENIVI Platform

2016-04-27 | 11:00

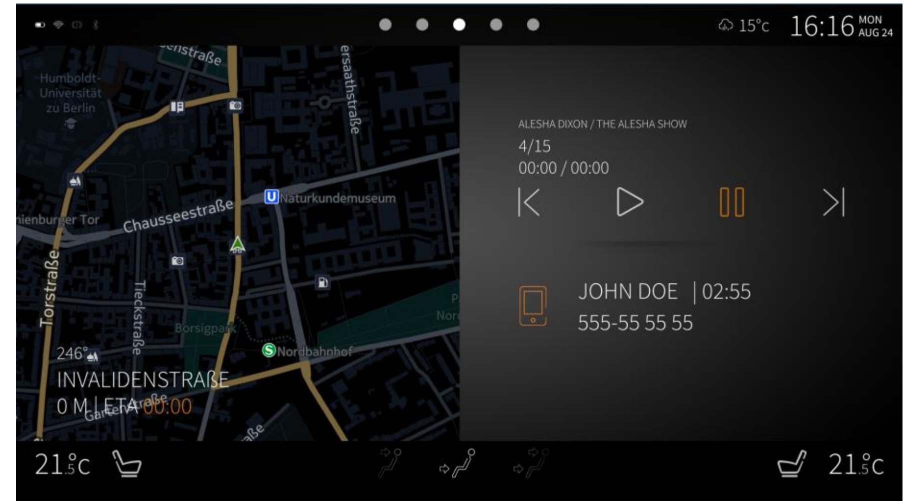
Johan Thelin
System Architect // EG HMI
Pelagicore

This work is licensed under a Creative Commons Attribution-Share Alike 4.0 (CC BY-SA 4.0)
GENIVI is a registered trademark of the GENIVI Alliance in the USA and other countries
Copyright © GENIVI Alliance 2016

Introduction

- An application centric platform based on Qt Automotive Suite running on a GENIVI platform

- We will look at code for the various parts as well as concepts and overviews



Why Applications?

- Life-cycle – innovation in consumer electronics move too fast for IVI, features need to be added during the life-cycle of a program
- Validation – partitioning the system into independent applications and a smaller platform reduces the validation work and variant configuration complexity
- Consumer Expectations – the customers are used to apps

 CC BY-SA 4.0

Why QtAS

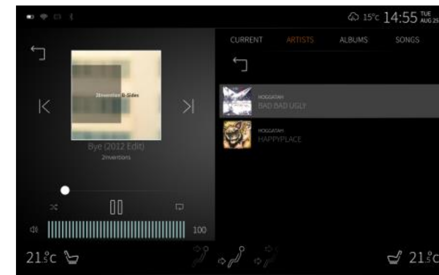
- Based on Qt, a mature toolkit used in multiple verticals
  - Internationalization
  - Support for left-to-right and right-to-left
  - Easy to implement UIs with a very varied appearance – full design freedom
  - Dynamic loading of UI parts – possible to control memory consumption
  - A clear model for building reusable components
  - Much, much, more

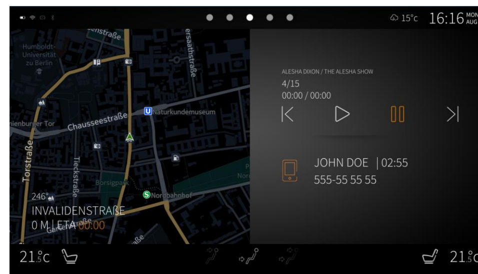
- QtAS is adapted and extended for IVI – Qt for an automotive architecture
- Takes Qt from a toolkit to a platform

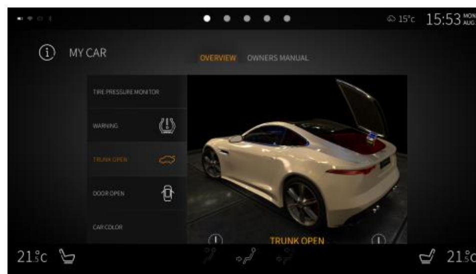
 PELAGICORE CC BY-SA 4.0

Neptune

- Will be shown during the showcase tonight
- Demonstrates a Qt Automotive Suite-based, application centric platform
- Converged head-unit and instrument cluster







 PELAGICORE CC BY-SA 4.0

Quick Poll

- Who is familiar with C++?
- Who is familiar with Qt using C++?
- Who is familiar with QML?

Qt in one slide

- C++ framework for cross platform application development
  - Windows, OS X, Unix/Linux, Android, iOS, embedded Linux, others
- Has been around for more than 20 years
- Dual licensed, open source and commercial
- The base for numerous successful open source projects
  - KDE, VLC, Subsurface, Wireshark, more

```cpp
class MyClass : public QObject {
    Q_OBJECT
    Q_PROPERTY(QString text READ text WRITE setText NOTIFY textChanged)
    Q_ENUM(MyEnum)
public:
    MyClass(QObject *parent=0);
    Q_INVOKABLE void pickAName();
    QString name() const;
    enum MyEnum { Foo, Bar, Baz };
public slots:
    void setName(const QString &name);
signals:
    void nameChanged(QString name);
    // ...
};
```

- QObject
- Signals / slots
- Properties
- Invokables
- Enums

- This is C++
- And introspectable at run-time

 CC BY-SA 4.0

Connections

- QObject instances become loosely coupled building blocks

```
connect(sender, &SenderType::signalName, destination, &DestinationType::slotName);
```

- Old style syntax

```
connect(sender, SIGNAL(signalName()), destination, SLOT(slotName()));
```

 CC BY-SA 4.0

QML

- Qt Meta Language
- Builds on the C++ introspection and Qt concepts
  – Signals, slots, properties, invokables
- Adds Javascript and a declarative approach
- Super easy to extend using C++

 CC BY-SA 4.0

```
import QtQuick 2.5

Rectangle {
    width: 360
    height: 360
    Text {
        anchors.centerIn: parent
        text: "Hello World"
    }
    MouseArea {
        anchors.fill: parent
        onClicked: {
            Qt.quit();
        }
    }
}
```

- Instantiation
- Bindings
- Events

 CC BY-SA 4.0

Item Models

- The QAbstractItemModel is a base class and interface for large collections of data
  - QAbstractListModel is a simplified API for lists

- Dynamic updates
  - Rows added/removed/moved, columns added/removed/moved, data changed, and so on
- Dynamic population
  - canFetchMore / fetchMore

- Handled from QML using the view – delegate – model setup

 CC BY-SA 4.0

```
ListView {
    anchors.fill: parent
    model: myModel
    delegate: myDelegate
}

Component {
    id: myDelegate
    GreenBox {
        width: 40
        height: 40
        text: index
    }
}
```

- View
- Model
- Delegate

 PELAGICORE CC BY-SA 4.0

More item models

- Views
  - Repeater
  - ListView
  - GridView
  - PathView
- Models
  - ListModel
  - XmlListModel
  - QSqlTableModel
  - QAbstractItemModel, which is the generic interface

More on QML

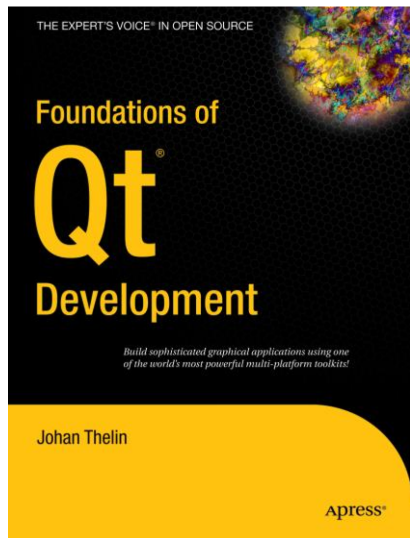
- Visual elements maps directly into an OpenGL scenegraph – great performance
- Easy to use access to low-level OpenGL features – shaders, particles
- Keeps a synchronized timeline allowing advanced animations

- Easy to extend OpenGL scenegraph to integrate other toolkits
  - The Foundry, Kanzi, NVIDIA UI Composer

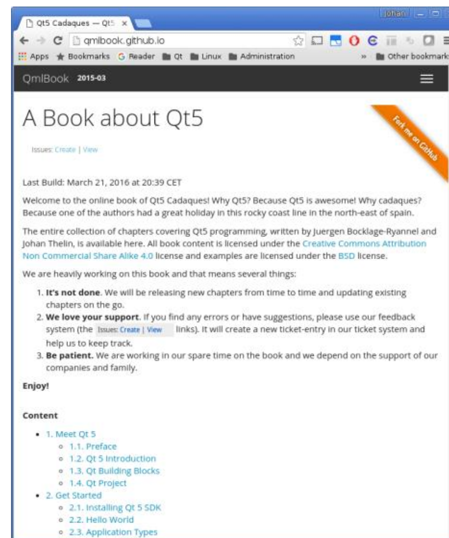
- Builds on Qt C++ – easy to extend with more C++ code
  - Handle complexity and performance

 PELAGICORE CC BY-SA 4.0

Learn More



About Qt and C++



www.qmlbook.org

 PELAGICORE CC BY-SA 4.0

The Beginning

 CC BY-SA 4.0

Application Manager

Wayland window compositor
- Wayland protocol compliant
- Token based display authorization for registered apps
- Implement in QML with full Qt animation support

App launcher
- Central point for starting and stopping internal and 3rd party apps
- Managing out-of-memory situations
- Quick launch for all Qt based apps

Application Manager

Security and Lifecycle Management
- Application isolation via Linux Containers
- Package installation, updates and removal using self contained bundles

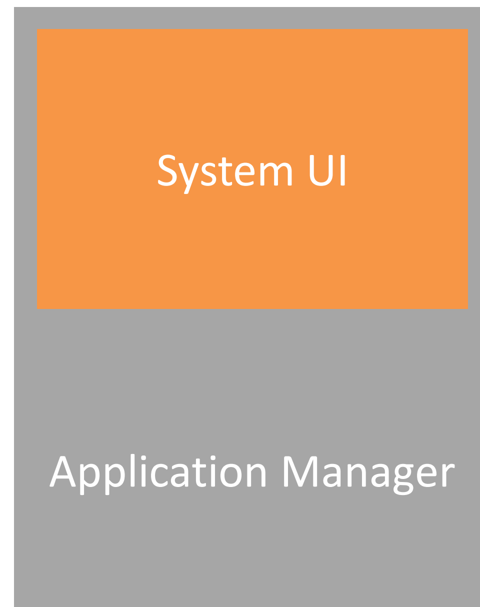
User input management
- Central virtual keyboard component
- Transparently used by all apps
- Integrated with Wayland compositor

 PELAGICORE CC BY-SA 4.0

Application Manager and System UI

- Application Manager provides the mechanisms, System UI the behavior

- Application Manager is the QML run-time environment in which the System UI is executed. Exposes the following APIs:
  - ApplicationManager, for launching, stopping and controlling applications
  - ApplicationInstaller, for installing, updating and removing applications
  - WindowManager, for implementing a Wayland compositor
  - NotificationManager, for implementing org.freedesktop.Notification



 PELAGICORE CC BY-SA 4.0

F.A.Q.

- Wayland is a protocol, Weston is a reference application
- Application Manager replaces Weston

- The System UI runs in the Application Manager process
- The System UI controls the compositor behavior
- The System UI is pure QML
  - Can be extended with C++, but does not have to

PELAGICORE CC BY-SA 4.0

```
import QtQuick 2.0
import io.qt.ApplicationManager 1.0

ListView {
    id: appList
    model: ApplicationManager
    delegate: Text {
        text: name + "(" + id + ")"
        MouseArea {
            anchors.fill: parent
            onClick: ApplicationManager.startApplication(id)
        }
    }
}
```

Privileges and Processes

unprivileged	root	unprivileged	sandboxed
	Installation Helper (root)	Runtime native	Native App
Application Manager	Runtime / Sandbox setup (root)	Runtime HTML	HTML App
		Runtime QML	QML App

 CC BY-SA 4.0

Back to the Example

- The application is being launched by Application Manager…

 PELAGICORE CC BY-SA 4.0

An Application and its Surroundings

Application

Qt

Qt IVI

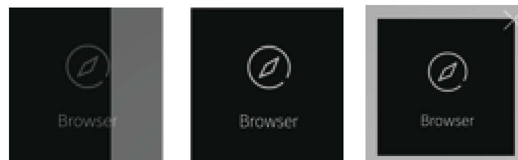
UI
Components

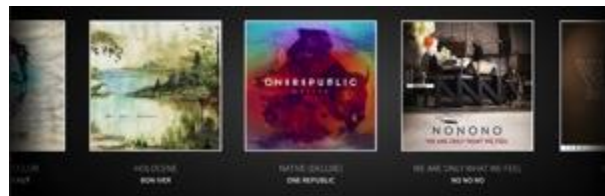
System UI

Application Manager

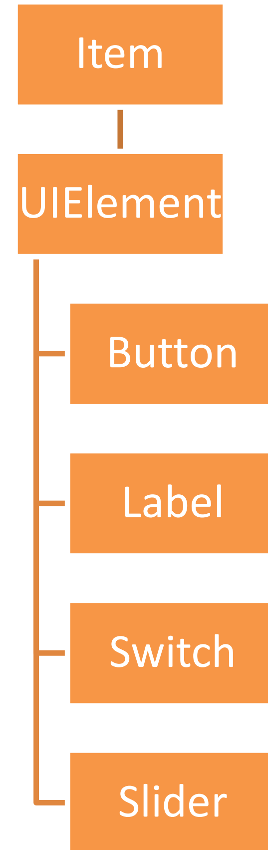
UI Components

- Application base-class
- Common elements
  - buttons, labels, sliders, lists…
- Common views
  - Supporting driver side, bidi…
- Common transitions
  - Animations, fade-in, fade-out…
- Combines graphics and behaviour



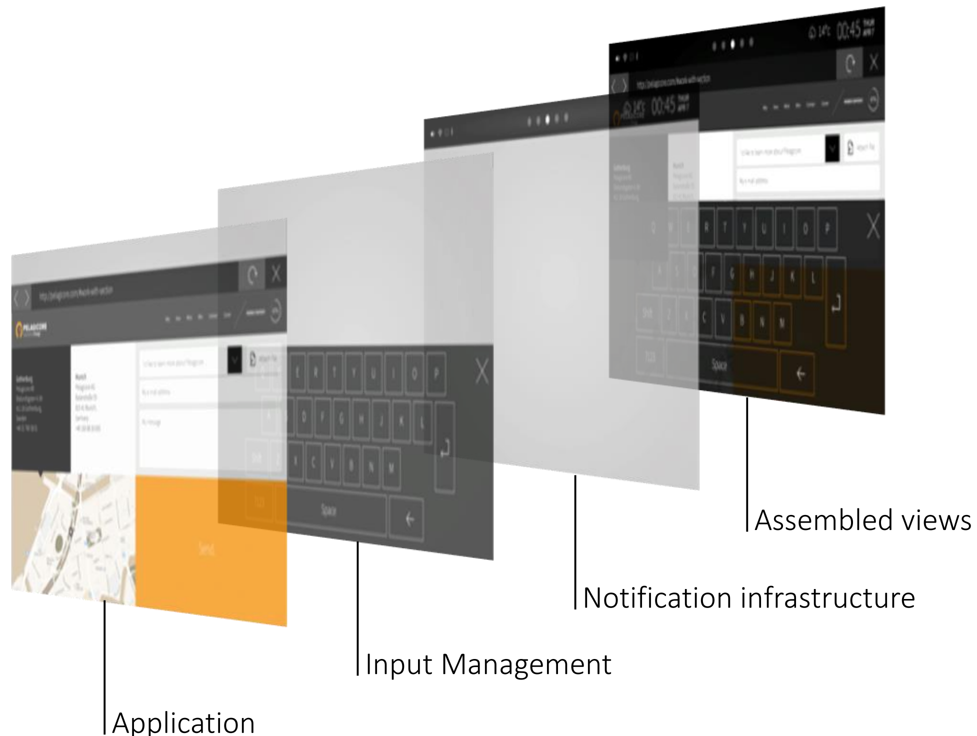


```
// Divider.qml
import QtQuick 2.1
import QtQuick.Layouts 1.0
import controls 1.0
import utils 1.0

UIElement {
    id: root
    hspan: 12

    Image {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        source: Style.gfx2("timeline")
    }
}
```



Compositing

- UI Components are combined onto a Wayland surface
- Application Manager picks it up and the System UI composes it into a screen
- Surfaces can be tagged using Wayland, i.e. I'm a popup, to allow the compositor to proper layout surfaces.



One compositor, including transition animations on a single slide.

```
import QtQuick 2.0
import io.qt.ApplicationManager 1.0

// simple solution for a full-screen setup
Item {
  id: fullscreenView

  MouseArea {
    // without this area, clicks would go "through" the surfaces
    id: filterMouseEventsForWindowContainer
    anchors.fill: parent
    enabled: false
  }

  Item {
    id: windowContainer
    anchors.fill: parent
    state: "closed"

    function surfaceItemReadyHandler(index, item) {
      filterMouseEventsForWindowContainer.enabled = true
      windowContainer.state = ""
      windowContainer.windowItem = item
      windowContainer.windowItemIndex = index
    }

    function surfaceItemClosingHandler(index, item) {
      if (item === windowContainer.windowItem) {
        // start close animation
        windowContainer.state = "closed"
      } else {
        // immediately close anything which is not handled by this container
        WindowManager.releaseSurfaceItem(index, item)
      }
    }

    function surfaceItemLostHandler(index, item) {
      if (windowContainer.windowItem === item) {
        windowContainer.windowItemIndex = -1
        windowContainer.windowItem = placeHolder
      }
    }

    Component.onCompleted: {
      WindowManager.surfaceItemReady.connect(surfaceItemReadyHandler)
      WindowManager.surfaceItemClosing.connect(surfaceItemClosingHandler)
      WindowManager.surfaceItemLost.connect(surfaceItemLostHandler)
    }

    property int windowItemIndex: -1
    property Item windowItem: placeHolder
    onWindowItemChanged: {
      windowItem.parent = windowContainer  // reset parent in any case
    }

    Item {
      id: placeHolder;
    }

    // a different syntax for 'anchors.fill: parent' due to the volatile nature of windowItem
    Binding { target: windowContainer.windowItem; property: "x"; value: windowContainer.x }
    Binding { target: windowContainer.windowItem; property: "y"; value: windowContainer.y }
    Binding { target: windowContainer.windowItem; property: "width"; value: windowContainer.width }
    Binding { target: windowContainer.windowItem; property: "height"; value: windowContainer.height }

    transitions: [
      Transition {
        to: "closed"
        SequentialAnimation {
          alwaysRunToEnd: true

          // your closing animation goes here
          // ...

          ScriptAction {
            script: {
              windowContainer.windowItem.visible = false;
              WindowManager.releaseSurfaceItem(windowContainer.windowItemIndex, windowContainer.windowItem);
              filterMouseEventsForWindowContainer.enabled = false;
            }
          }
        }
      },
      Transition {
        from: "closed"
        SequentialAnimation {
          alwaysRunToEnd: true

          // your opening animation goes here
          // ...
        }
      }
    ]
  }
}
```

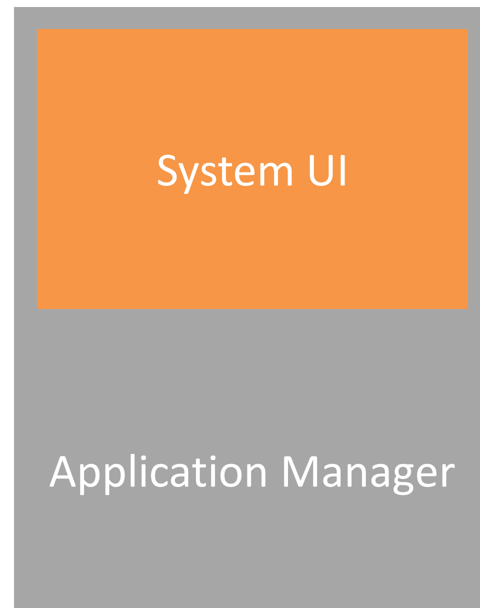
Sorry about the readability…

```
// Connect to signals
Component.onCompleted: {
    WindowManager.surfaceItemReady.connect(surfaceItemReadyHandler)
    WindowManager.surfaceItemClosing.connect(surfaceItemClosingHandler)
    WindowManager.surfaceItemLost.connect(surfaceItemLostHandler)
}

// Handle new surfaces (place in container)
 function surfaceItemReadyHandler(index, item) {
    filterMouseEventsForWindowContainer.enabled = true
    windowContainer.state = ""
    windowContainer.windowItem = item
    windowContainer.windowItemIndex = index
}

// Find App instance in ApplicationManager from surface
var appIdForWindow = WindowManager.get(winIndex).applicationId
```

 PELAGICORE CC BY-SA 4.0

Application Manager and System UI

- The System UI is a QML script executed inside the Application Manager
- Application Manager provides mechanisms, the System UI implements the OEM specific behavior
- The System UI is built from UI Components and custom parts, just like any other app in the system



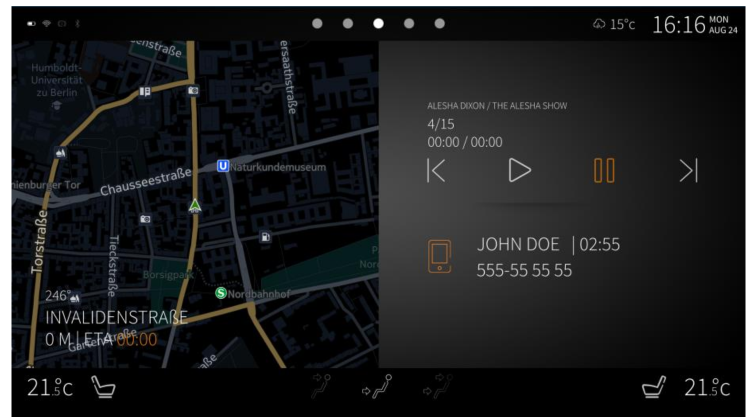
 CC BY-SA 4.0

```
// Client side, i.e. App
ApplicationManagerWindow {
  Component.onCompleted: { setWindowProperty("winType", "main") }
}

// Server side, i.e. System UI
Connections {
  target: WindowManager
  onSurfaceItemReady: {
    if (WindowManager.surfaceWindowProperty(item, "winType") === "main") { /* it's a "main" */ }
    if (WindowManager.get(index).id === "com.pelagicore.nav") { /* coming from the nav application */ }
}

// Another approach is to use the window title as tag
```

 PELAGICORE CC BY-SA 4.0

System UI

- Typically integrates
  - Virtual keyboard and handwriting areas
  - Notifications
  - Popups
  - Launcher
  - Central settings
- Coordinates with other sub-systems
  - NSM
  - LUC preservation / restoration
  - and more…



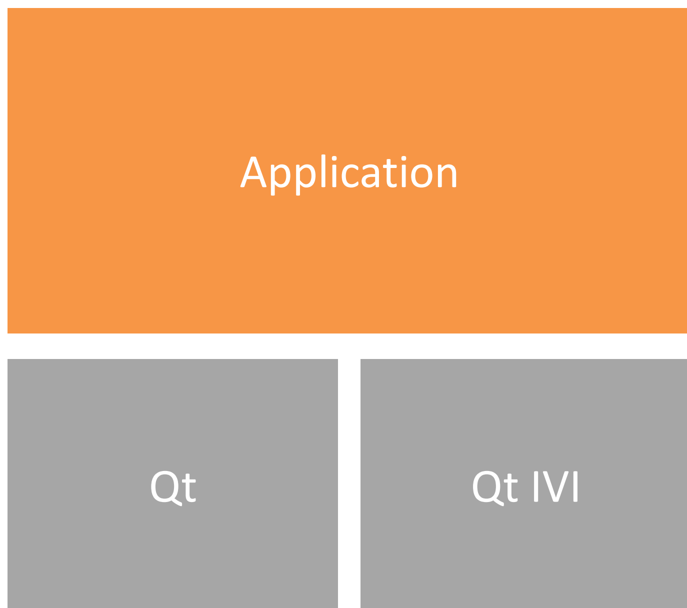
 PELAGICORE CC BY-SA 4.0

Application Processes

- Application Manager supports in-process apps and out-of-process apps
- In process apps are great for
  - Startup
  - Development
  - Hardware where Wayland support is missing
- Out of process apps are great for
  - Fully decoupled
  - Can easily be containerised
- Application Manager supports mixed mode setups

 PELAGICORE CC BY-SA 4.0

Back to our App



- The application uses a number of interfaces

- Qt
- Qt IVI
- Anything else… no technical limitations

QtGeniviExtras

- Uses Qt interfaces for common platform services
- A part of the QtIVI module

- Current
  - Qt Categorized Logging integrated on top of DLT

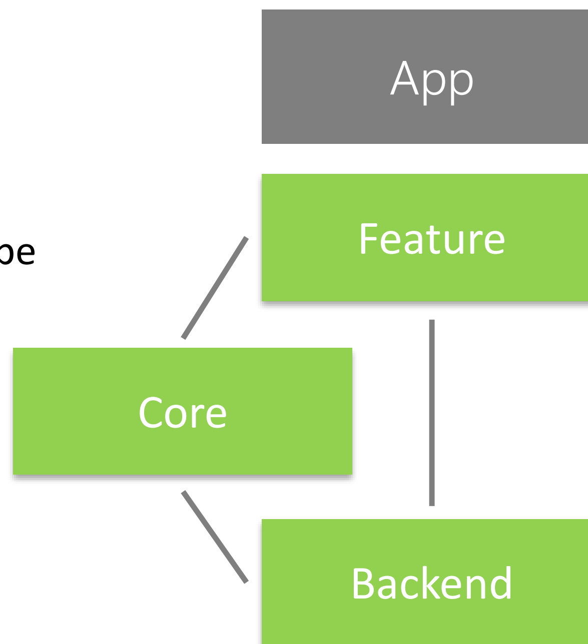
- On the road-map
  - QSettings + support integrated on top of PCL
  - NSM integration

 PELAGICORE CC BY-SA 4.0

Qt IVI

- Extensible Qt APIs
  - A pattern for how to extend Qt with new interfaces
  - Base classes to add structure
  - Reference implementation of APIs based on W3C scope

- Separation of Frontend and Backend
- Multiple backends for various use-cases
  - Different sub-systems
  - Simulation
  - Unit-tests



 PELAGICORE CC BY-SA 4.0

Qt IVI - Frontends

- APIs for app development

- Bindable QML interfaces, easy to use C++ interfaces
- Common development experience regardless of backend
  - Same error messages
  - Same query language for searches
  - Agnostic to backend implementation
    - In process code, e.g. shared object
    - IPC, e.g. d-bus, socket, CommonAPI C++
    - Networking, e.g. TCP/IP, CAN, MOST

Qt IVI - Backends

- Implements an interface defined by the frontend according to guidelines
  - Asynchronous
  - Stateless
  - Etc
- All backends share a set of key unit tests
  - Sequence order
  - Out of range handling
- We use the compiler and unit tests to ensure that the behavior is the same for all

 PELAGICORE CC BY-SA 4.0

Qt IVI - Bindable Interfaces, Optimistic UIs

- Having a bindable interface is something the QML like, no app logic needed to connect to a backend when it becomes available, just wait for the available property to go true

- The bindable interface, i.e. the frontend, supports optimistic Uis, i.e. safe defaults
  - Not always what you want, but great when used correctly

```
ClimateControl { // Interface
    id: climateControl
    discoveryMode: ClimateControl.AutoDiscovery
}

CheckBox { // Usage
    text: "Air Recirculation"
    checked: climateControl.airRecirculation.value
    enabled: climateControl.airRecirculation.available
    onClicked: {
        climateControl.airRecirculation.value = checked
    }
}
```

 CC BY-SA 4.0

Typical Platform APIs

- The APIs provided to apps in an IVI system are always extended
- Everyone adds something unique

Embedded Linux | GENIVI | OEM

 PELAGICORE CC BY-SA 4.0

Qt IVI and Franca IDL

- We can code generate wrapper QObjects from Franca IDL
  - The result is usually not very nice from QML – the Qt feeling is missing
- We need to map high level concepts
  - QAbstractItemModel – large lists
  - Naming – how can we agree or map names correctly
  - Can we use Franca IDL to define default values
  - Probably more in Qt and in other toolkits as well
- Defining guidelines will make the GENIVI APIs extendible
- Let’s talk in the Application Framework Working Session on Thursday!

Goal Architecture v0.9

Managed Applications — "Apps", e.g. Commercial Music Services, Weather, Social Networks...

Native Applications — E.g. Vehicle Functions, Climate (HVAC), Navigation, Radio ...

System User Interface

Application Manager | Web App Runtime | Java App Runtime | Prog. framework/abstraction (Qt and others)

Business Logic / Platform Adaptions (optional, dep. on circumstance)

Radio & Tuners: AM/FM, DAB/DRM, Broadcast Data services, SDARS, Terres-trial TV, HD Radio, TMC/VICS

Telephony: Telephony Stack (eg.Ofono)

Navigation/LBS: Traffic Info, Navigation Core, Map Viewer, Map Data Service, Positioning, POI Mgr

Media Framework: Playback Control, Browser, Indexer, Music Identi-fication

Media Sources: USB Mass Storage, Commercial Streaming, Bluetooth Stream, MTP, Internet Radio, AUX, DLNA, iAP

Internet Functions: DUMM, Web Browser, Cloud Based Services

Bluetooth: Mess-aging, Phone Book, Bluetooth Stack (eg.Bluez), Hands-free, Media Playback, Tethering

Camera Functions: Rear View Camera, Guidance / Overlay

Speech: Speech Input (ASR), Speech Output (TTS), Speech Dialog, Speech to Text Dictation

HMI Support: I18N & L10N, Graphical Framework, Pop-Up Mgr, Buttons, Hand-writing

CE Device Integration: Smart Device Link, CarPlay™, Android Auto, MirrorLink

PIM: Shared Address Book, Internet Account Manager, Device Sync, Calendar, Internet Account Sync

Vehicle Interface: Seat Heating, Climate Control, Vehicle Settings, Vehicle Interface API(Eg.AMB)

Device Mgmt: Advanced Handover Support, uevent / udev

Audio Mgmt: Audio Manager, Pulse Audio

Audio/Video Processing: EC/NR, Alsa, Video Inputs (i.e. V4L), SRC, Codecs, Gstreamer

Graphics Support: Layer Manage-ment, OpenGL (EGL), IVI Compositor (Wayland Protocol)

Network Mgmt: ConnMan, Traffic Shaping, Firewall Rule Mgmt

Networks: EAVB, SOME-IP, Vehicle Bus Proxy (CAN, FlexRay), Wifi, Teth-ering, NFC, INC, ICC

IPC: DBUS, CommonAPI Runtime, Message Broker/Routers

Persistence: Persistence Client Lib, Pers. Admin, SQLite, Custom storage, Pers. Health Monitor

SW Management: SW Loading Mgr, Package Mgr, Module Loader, SOTA Client

Lifecycle: Node State Mgr, Node Startup Controller, Node Resource Mgr, Node Health Monitor

User Mgmt: User Identification, User Switch, User Data Migration

Housekeeping: Error/Event Logging(DLT), Exception Handling, Statistics, Coding / System Config.

Security Infrastructure: HSM, Encryption, Signatures, MAC, Anomaly Detection

Diagnostics: UDS, DTCs, Automotive Diagnostics, Remote Diagnostics

Generic libraries (libc, etc.) | Low-level system libraries (libusb etc.)

Drivers, BSP, Linux Kernel

Initial Bootloader

NOTE:

The block diagram describes only subsystems and functional blocks within them, but a full design may break these down further into detailed software components.

Therefore the color coding can only describe an approximation, i.e. what the majority of content in this box may become.

There will be individual exceptions where it is not feasible or desired for a particular software component.

Legend

- **Placeholder** Component (Requirements)
- **Abstract** Component (Interfaces defined)
- **Specific** Component (Implementation)
- **Not specified**

Access Control

App

Manifest

Bundle

Signed

Application Manager Installer

Installed

Application Manager Runtime

Executed in limited environment

 PELAGICORE CC BY-SA 4.0

Application Manifest

- Based on YAML
  - Readable
  - Easy to write
  - Can be commented
- Contains
  - Meta data for launcher
  - Info about access rights – *Capabilities*
  - Extendable


```yaml
formatVersion: 1
formatType: am-application
---
id:      'com.pelagicore.movies'
icon:    'icon.png'
code:    'Movies.qml'
runtime: 'qml'
name:
  en: 'Movies'
  de: 'Filme'

categories: [ 'app' ]
built-in: yes

capabilities: ['video', 'sound', 'storage', 'network' ]
```

Application Bundles

- Signed
  - Trusted source independent of distribution mechanism
- No dependencies
  - No dependency hell
  - No app-to-app security issues
  - The only version compatibility to handle is app vs platform
- No installation scripts
  - No tricky security situations – app is extracted into a single read-only directory

 CC BY-SA 4.0

System Access Limitation

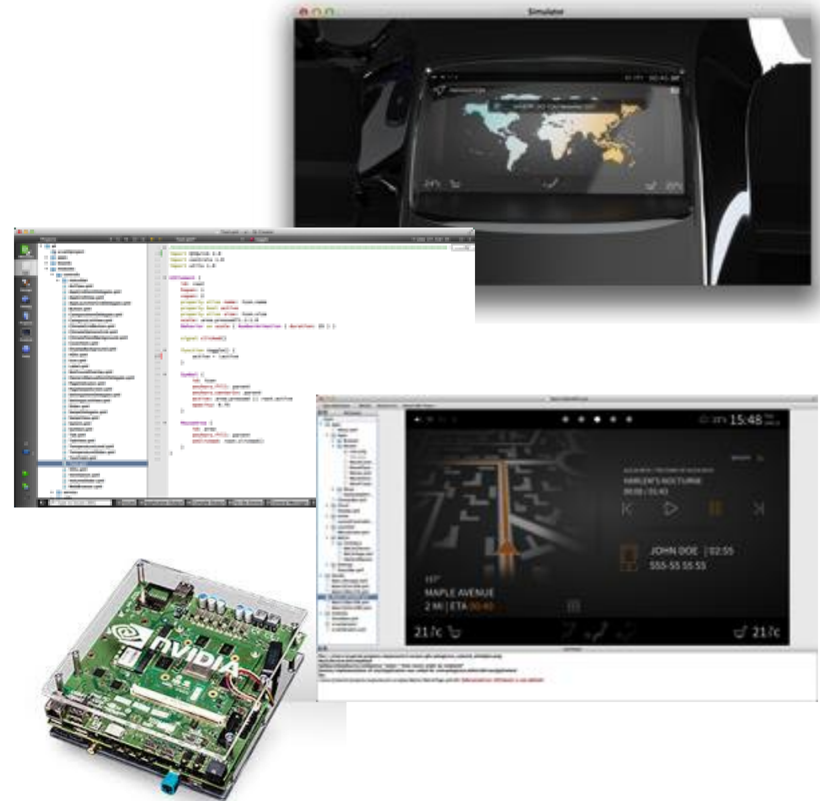
- Application Manager executes applications in different execution environments depending on the level of trust, e.g.
  - Native applications may run uncontained as an ordinary user
  - Some applications may run inside an UID jail
  - Some applications may run inside a container-based sandbox
    - Pelagicontain, based on LXC and platform gateways
  - More mechanisms through plugins

- Application Manager can use these systems in parallel, e.g.
  - Tuner and Phone are not contained, runs at user privilege level
  - Webbrowser runs inside a sandbox, cannot see anything but the strictly needed
  - Spotify runs inside a UID jail, can do less, but still sees large parts of the system

 CC BY-SA 4.0

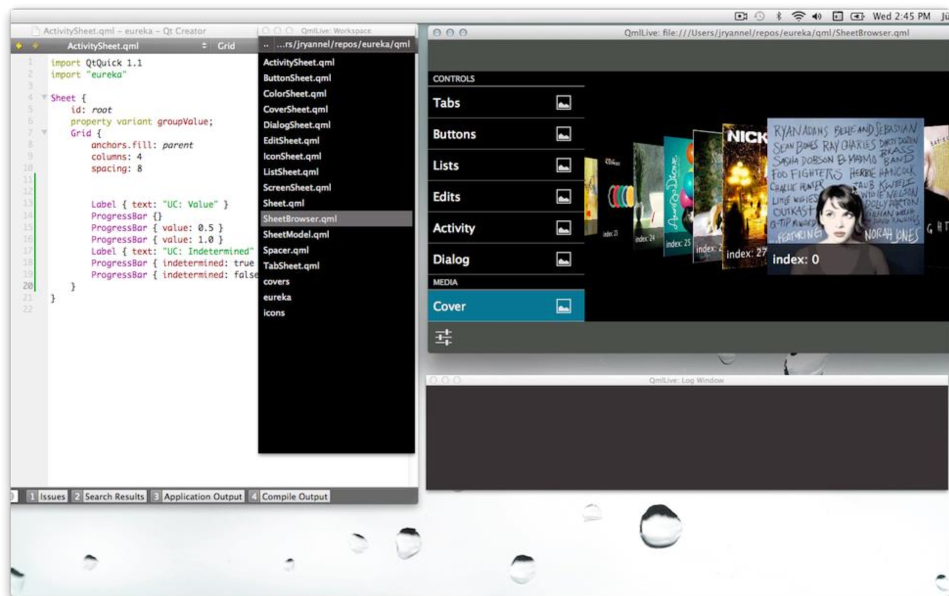
Application SDK

- Qt Creator based – supports Windows/OSX/Linux
- Integrated with your System UI and UI Components
- QtAS.QmlLive – enables quick round-trip to target hardware
- Qt IVI Simulator – enables evaluation on desktop against simulated service APIs
- Reference UI – provides a starting point



QmlLive

- Rapid UI prototyping tool
- Designer friendly
- A server / client automatic reloader tool
  - Makes it possible for designers to test out designs on the actual target



 PELAGICORE CC BY-SA 4.0

Summary

- Qt Automotive Suite – http://www.qt.io
- QmlLive – https://github.com/Pelagicore/qmllive
- QmlBook – http://qmlbook.org
- Pelagicore Labs – http://labs.pelagicore.com

 PELAGICORE CC BY-SA 4.0

CC BY-SA 4.0