Qt for iOS A to Z

Mike Krus, Senior Software Engineer at KDAB

Qt WORLD SUMMIT 2015

- iOS Hardware & Software platform
- Setting up Qt
- Your first project
- qmake for iOS
- QML for iOS
- Using native frameworks
- CMake support
- Building libraries
- Putting it all together

- iPhone, 4S and up
- iPad 2 and up, iPad Air, iPad Mini
- All iOS devices run ARM
- Graphics is PowerVR
- Full complement of sensors
  - GPS, magnetometer, tilt, etc
- Screen resolutions vary
  - Retina display
- Modern fast CPU, GPU
- Back to the 80s constraints for memory, battery...

- iOS 8 and up (for Qt 5.5)
- Low-level system shared with OS X
  - Mach microkernel
  - BSD userspace libraries (libC, crypto)
- Apple system libraries
  - CoreAudio, CoreAnimation, CoreLocation, AVFoundation, CoreWLAN
- Cocoa Touch
  - Objective-C, like Cocoa
  - Natively designed for mobile & touch
  - UIKit, Standard widgets
  - `UI` class prefix
  - Many others (MapKit, HealthKit, ...)

- UIKit provides standard high-level view structures
  - Paged, tabbed, master / detail
  - User and designers familiar with the behaviour and visual presentation
- Also supports basic navigation and transitions
  - Push / Pop navigation
  - Modal presentation
- Absence of pop-up / pop-over UI
  - Modal yes / no dialogs the (infrequent) exception
- Transitory modal UI for rotary wheels
- Powerful animation engine
- Users will abandon apps they can't interact with

- A Mac
  - Basic laptop or Mac Mini sufficient
  - Mac Mini works as headless build machine
  - Not a VM on Not a Mac
- An Apple Id
- An Apple developer subscription
  - Applies to more than one Apple ID
  - No limit on team size
  - iOS 9 introduce 'Personal Teams'
- Xcode

- Don't be afraid of Xcode
- Entrypoint to dedicated editors
- Manages developer account
- Manages devices and applications
- Builds & deploys your code
- Archive management
- SDK documentation and examples
- Excellent debugger

The simulator is *not* an emulator

- Application compiled for x86 or x64
- Links against custom frameworks providing iOS APIs
- Simulator frontend allows hardware emulation
- Runtime environment
- Differences from real hardware
  - Code-signing, OpenGL performance, sensors

- **You must run your app on real hardware**
- iOS applications must *always* be **code signed**
  - Certifies origin of the app
  - Prevents tampering
- Deployment requires **Provisioning Profiles**
  - Enables some features, iCloud, Health, Push Notifications...
  - Controls where app can run
  - Developer profiles
  - AppStore or Ad-Hoc profiles
- Xcode is your friend
- **You must run your app on real hardware**

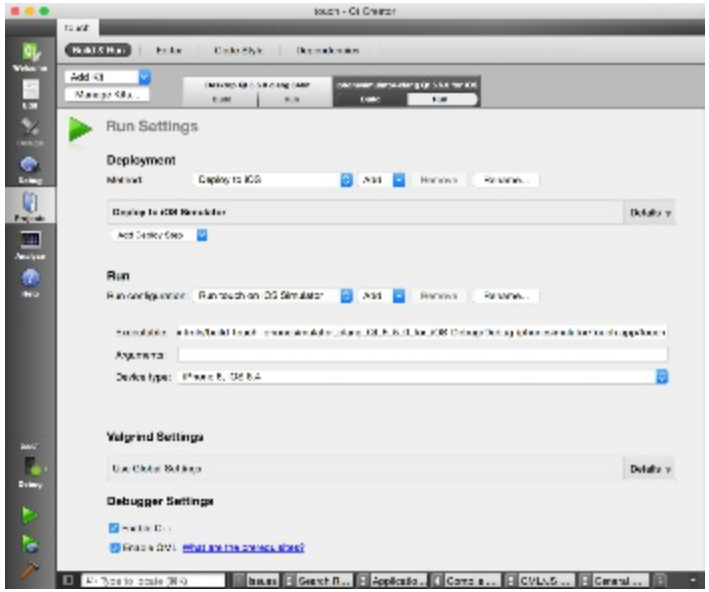
- iOS never exposes the filesystem to the user
- At runtime, applications can only access files within their 'sandbox'
  - Implemented as kernel access control (ACL), not a chroot
- No access to other application's data
- Other restrictions
  - No use of private APIs
    - Checked mechanically by tooling
  - No downloading code or JIT-ing
    - Attempts to mark data as executable will fail
- Application groups in iOS 8
  - Apps with matching group IDs can share containers
  - Group IDs defined in the member center, registered in the provisioning

- Everything about your app
  - App Store listing, price, In-App purchase
  - Manage your versions, builds
- Test Flight
  - Register up to 1000 testers (iOS 8 or later)

- Recent version of Qt & Creator
  - Qt 5.4
  - Static build
- non-GUI layers of Qt compile as normal
  - Unix / Mach backend for files, sockets, memory
- QPA layers maps `QWindow` to `UIWindow`
- Widget-based UI possible
  - Not recommended
- QtQuick UI rendered using OpenGL
- All the UIKit controller / interface machinery is ignored

KDAB

Time to try Creator...



KDAB

`qmake` generates

- Makefiles
- `MyProject.xcodeproj`
- `Info.plist`
- qml files, contained in a qrc
- `main.cpp`
- `myproject_plugin_import.cpp`
- `myproject_qml_plugin_import.cpp`

- Application is a *bundle*
  - Specially structured directory
    - *Documents, Library/Cache...*
  - Not a compressed zip/jar
- Bundles contain an `Info.plist` XML file
  - Metadata about author, supported platforms, architectures, copyright
  - Specifies executable to launch inside the bundle
  - File-types / URLs / mime-types
- Bundle has a unique identifier (reverse DNS style), version number, build number
- Also contain resources as plain files

- `mac vs osx vs ios`
- Variables

```
1  QMAKE_IOS_DEPLOYMENT_TARGET = 8.0
2  QMAKE_IOS_TARGETED_DEVICE_FAMILY = 2
3  QMAKE_IOS_DEVICE_ARCHS = armv7 arm64
4  QMAKE_IOS_SIMULATOR_ARCHS = i386 x86_64
5
6  VERSION = 1.2.3
7  BUILDID = 55
8
9  plist.input = Info.plist.in
10 plist.output = $$OUT_PWD/Info.plist
11 QMAKE_SUBSTITUTES += plist
12 QMAKE_INFO_PLIST = $$OUT_PWD/Info.plist
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.d
<plist version="1.0">
<dict>
    <key>CFBundleIdentifier</key>
    <string>com.kdab.${PRODUCT_NAME:rfc1034identifier}</string>
    <key>CFBundleDisplayName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundleName</key>
    <string>${PRODUCT_NAME}</string>
    <key>CFBundleShortVersionString</key>
    <string>$$VERSION</string>
    <key>CFBundleVersion</key>
    <string>$$VERSION.$$BUILDID</string>
    <key>LSRequiresIPhoneOS</key>
    <true/>
    <key>UISupportedInterfaceOrientations</key>
    <array>
        <string>UIInterfaceOrientationLandscapeLeft</string>
        <string>UIInterfaceOrientationLandscapeRight</string>
    </array>
    <key>UIFileSharingEnabled</key>
    <true/>
    ...
</dict>
</plist>
```

- Include *properly named files* in your bundle

```
meta.files = $$files($$PWD/meta/*)
QMAKE_BUNDLE_DATA += meta
```

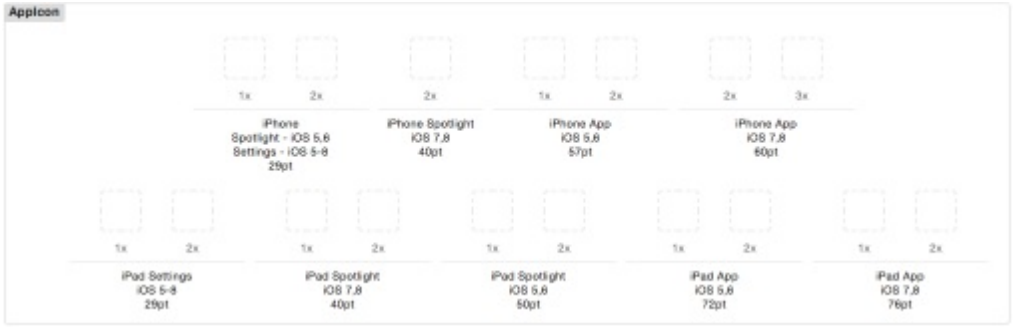
- Reference in `Info.plist`

```
...
<key>CFBundleIcons</key>
<dict>
    <key>CFBundlePrimaryIcon</key>
    <dict>
        <key>CFBundleIconFiles</key>
        <array>
            <string>Icon-57.png</string>
            <string>Icon-72.png</string>
            <string>Icon-72@2x.png</string>
        </array>
        <key>UIPrerenderedIcon</key>
        <true/>
    </dict>
</dict>
...
```

- Include *asset bundles* in your bundle

```
meta.files = $$files($$PWD/meta/*.xcassets)
QMAKE_BUNDLE_DATA += meta
```

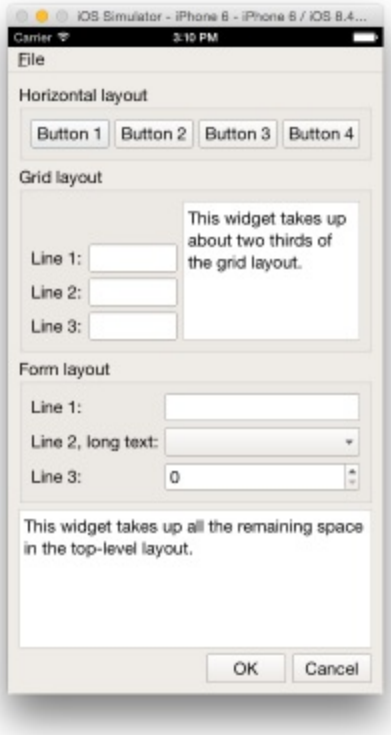
- Edit in Xcode



Remember you're working on a copy!

KDAB

Widgets work, but...



... don't

- QML
  - UI rendered using OpenGL
  - Designer friendly
  - Animations
- Quick.Controls, Quick.Dialogs
  - More layouts
  - `ComboBox`
  - `StackView`
  - `FileDialog`
    (`fileDialog.folder: fileDialog.shortcuts.pictures`)
  - Text Input
    - Selection is different
    - Keyboard avoidance is a pain (see `Qt.inputMethod.hide()`)

- Retina & resources
  - Coordinates are in `qreal`, *points NOT pixels*
  - Set `border.pixelAligned: false`
- Image sources support `@2x`

```
Image { src: "/images/foo.png"; width: 200; height 200 }
```

- If present, will load `foo@2x.png` (400x400 px) on retina devices
  - Does not support `@3x` for iPhone 6+
- Embrace this, don't fight it by playing with size and scale of root item!

- Plain QML has no style
- Controls Look like widgets
- Native appearance is problematic
  - Define native!
  - Moving target: Apple restyle UIKit across OS upgrades

Qt Demo /Users/Shared/Qt/Examples/Qt-5.5/quick/controls/touch

- Missing canonical UI elements
  - Toolbar, buttons, ...
  - Navigation bar, `UINavigationController`, modal views...

- Hard to not load everything at start up
- `Loader` useful but destructive
- Using components

```
1  var component = Qt.createComponent("foo.qml")
2  if (component.status == Component.Ready) {
3      newView = component.createObject(parent, {opacity: 0, "anchors.fill": parent})
4
5      previousView = currentView
6      currentView = newView
7      if (previousView) {
8          previousView.viewWillDisappear()
9          fadeOut.target = previousView
10         fadeOut.running = true
11     }
12     currentView.viewWillAppear()
13     fadeIn.target = currentView
14     fadeIn.running = true
15 }
```

- Build you own sets of components
- Even with Controls, style is applied to instances

- Sensors
- Camera
- Position (GPS), Location (Maps)
- In-App purchase
- Qt3D
- ...

- Coverage not complete
  - File-type association
  - Background operations
  - Sharing
  - iCloud
  - Access to contacts, accounts (Twitter / Facebook)
  - CoreMotion (steps...), CoreLocation, MapKit, HealthKit, GameKit, PassKit, Security, PushKit...
  - WebKit!
  - ...
- Through specialised Objective-C APIs
- Not wrapped by Qt, easy to invoke yourself

- Objective-C & C++ combined
- Language syntaxes are orthogonal
- `.mm` file extension
- `OBJECTIVE_SOURCES` in `qmake`

Bridging the Objective-C and Qt object systems is possible. Potentially relevant if incorporating a 3rd-party framework for iOS.

- Derive from `QQuickItem`
- Handle `windowChanged(QQuickWindow*)` signal to create `UIView`
- Handle `visibleChanged()` signal to show/hide
- Overload `geometryChanged(const QRectF &newGeometry, const QRectF &oldGeometry)` to resize

Demo ios/UITextView

- QPA layer exposes native resources
- Parent UIView hierarchy to main Qt window
- Unsuitable for embedding individual widgets

```
QT += gui-private
```

```
1  QWindow* window = ... ;
2  UIView *view = static_cast<UIView*>(QGuiApplication::platformNativeInterface()->
3      nativeResourceForWindow("uiview", window));
4  Q_ASSERT(view);
5
6  UIViewController* controller = [[view window] rootViewController];
7  Q_ASSERT(controller);
```

- Create a view / controller heirarchy programmatically
- Via a system or 3rdparty library

Use Objective C categories

```
1  @interface QIOSApplicationDelegate
2  @end
3
4  @interface QIOSApplicationDelegate(MyApp)
5  @end
6
7  @implementation QIOSApplicationDelegate(MyApp)
8  - (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
9  {
10     qDebug() << "Lets release some memory before we get killed";
11 }
12
13 - (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
14 {
15     ...
16 }
```

Use own delegate

```
1  @interface MyAppDelegate : UIResponder <UIApplicationDelegate, DBSessionDelegate>
2  +(MyAppDelegate *)sharedAppDelegate;
3  @end
4
5  @implementation MyAppDelegate
6  static MyAppDelegate *sharedAppDelegate = nil;
7  +(MyAppDelegate *)sharedAppDelegate {
8      static MyAppDelegate *shared = nil;
9      if (!shared)
10         shared = [[MyAppDelegate alloc] init];
11     return shared;
12 }
13
14 - (BOOL)application:(UIApplication *)application
15         willFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
16     return YES;
17 }
18 @end
19
20 int main(int argc, char *argv[]) {
21     ...
22     [[UIApplication sharedApplication] setDelegate:[MyAppDelegate sharedAppDelegate]];
23     ...
24 }
```

- Use `AUTO_MOC`, `AUTORCC`
- Static plugins issue
  - Don't have qmake build system
  - Wrap `qmlimportscanner`

```
1  cmake_minimum_required(VERSION 2.8.5)
2  project(QtWSToday)
3
4  SET(VERSION 0.1.0)
5  SET(BUILDID 0.1.0.55)
6
7  set(CMAKE_OSX_SYSROOT iphoneos8.4)
8  set(CMAKE_OSX_ARCHITECTURES "armv7;arm64;i386")
9  set(CMAKE_XCODE_EFFECTIVE_PLATFORMS "-iphoneos;-iphonesimulator")
10 set(CMAKE_AUTOMOC ON)
11 set(CMAKE_INCLUDE_CURRENT_DIR ON)
12 set(CMAKE_FIND_FRAMEWORK FIRST)
13
14 set(CMAKE_PREFIX_PATH "${CMAKE_PREFIX_PATH};/Users/Shared/Qt/5.5/ios/lib/cmake")
15 find_package(Qt5 COMPONENTS Qml Quick Core Network)
```

```
SET(PRODUCT_NAME MyQtApp)
set(SOURCES main.cpp qml.qrc Info.plist.in)
add_executable(MyQtApp MACOSX_BUNDLE ${SOURCES})
qt5_use_modules(QtWSToday Core Qml Quick Core Network)
target_link_libraries(QtWSToday "-framework QtWS -framework Foundation -framework Security"
    "-framework UIKit -framework CoreGraphics -lz"
    ${_qt5Quick_install_prefix}/lib/libqtpcre.a ${_qt5Quick_install_prefix}/lib/libqtharfbuzzng.a
    ${_qt5Quick_install_prefix}/plugins/qmltooling/libqmldbg_tcp.a
)

set_target_properties(QtWSToday PROPERTIES
    MACOSX_BUNDLE_GUI_IDENTIFIER "com.kdab.MyQtApp"
    MACOSX_BUNDLE_INFO_PLIST ${CMAKE_CURRENT_SOURCE_DIR}/Info.plist.in
    RESOURCE "${RESOURCES}"
    XCODE_ATTRIBUTE_CODE_SIGN_IDENTITY "iPhone Developer"
    XCODE_ATTRIBUTE_DEBUG_INFORMATION_FORMAT "dwarf-with-dsym"
    XCODE_ATTRIBUTE_GCC_PRECOMPILE_PREFIX_HEADER YES
    XCODE_ATTRIBUTE_GCC_PREFIX_HEADER ${CMAKE_CURRENT_LIST_DIR}/myqtapp.pch
    XCODE_ATTRIBUTE_INFOPLIST_PREPROCESS YES
    XCODE_ATTRIBUTE_IPHONEOS_DEPLOYMENT_TARGET 8.0
    AUTORCC ON
)
```

- Can build shared libraries (frameworks!) since iOS 8

```
1 TEMPLATE = lib
2 CONFIG += lib_bundle dll
3
4 FRAMEWORK_HEADERS.version = Versions
5 FRAMEWORK_HEADERS.files = libqtws.h
6 FRAMEWORK_HEADERS.path = Headers
7 QMAKE_BUNDLE_DATA += FRAMEWORK_HEADERS
```

- Building app extensions
  - `qmake` and `cmake` not up to it
  - Need to build a pure Xcode solution

DEMO

Thank you!

www.kdab.com

mike.krus@kdab.com