The Qt Safe Renderer

Rendering Safely
Animating Safely

Becky Worledge, The Qt Company

Agenda

- Functional Safety
- Soup
- Safe QML
- Controlling your Safe QML from Safe C++
- Safe Animations

Safety. Functional Safety.

- How do you make safe software that end users can rely on?
- Write to a robust code standard
- Have your code reviewed by knowledgeable people
- Kind of the same as any other peer-reviewed software?

Functional. Safety.

- In the world of Functional Safety, there are various coding standards:
  - MISRA
  - JSF
  - ASIL
  - Etc
- And there are various certifying bodies:
  - TÜV
  - exida
  - Etc
- And there are functional safety accreditation bodies
  - UKAS, IAF, ILAC
  - These chaps certify the certifiers that certify your code

How does Qt measure up to Functional Safety standards?

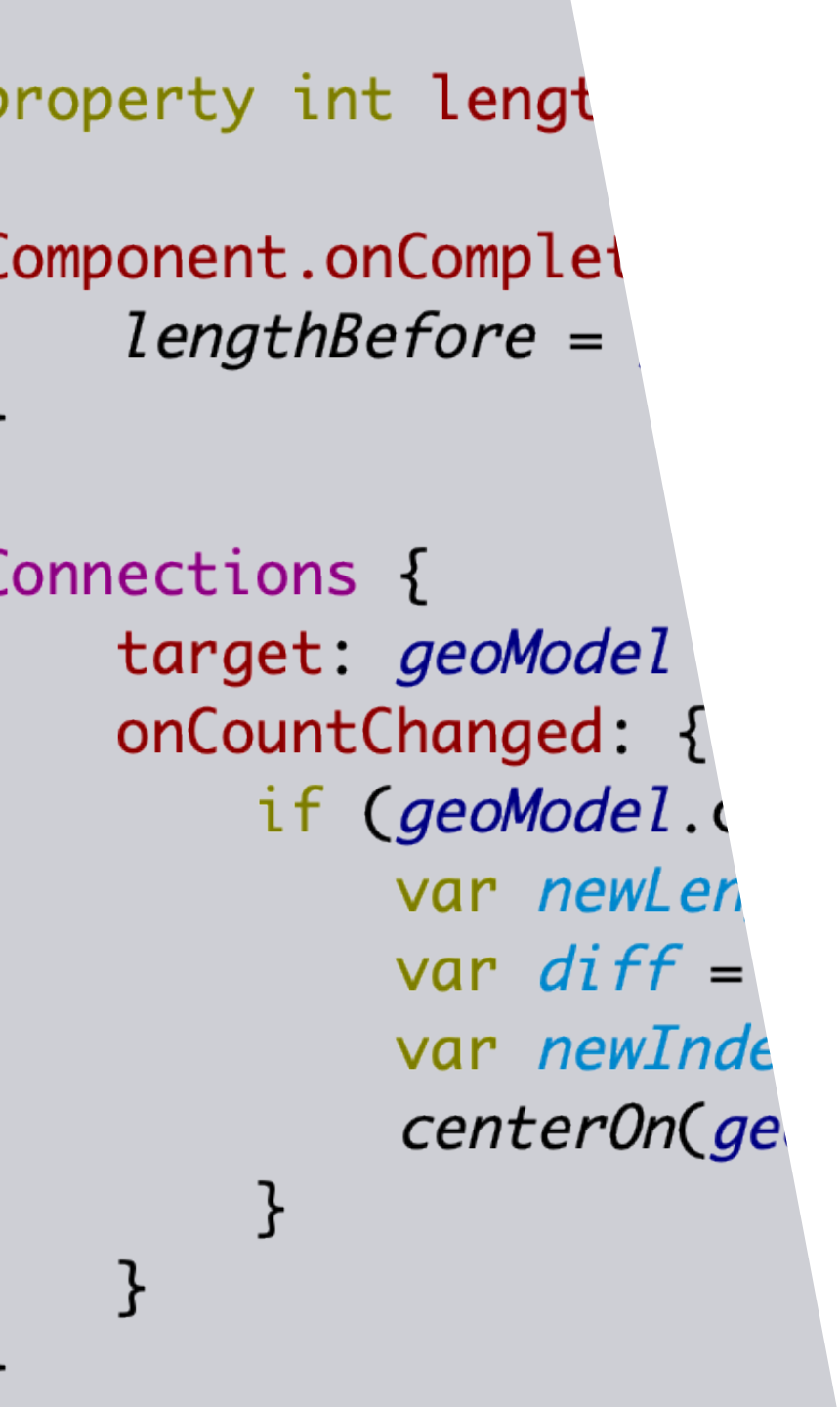
- Actually Quite Badly

Software of Unknown Provenance

- Qt is S.O.U.P.
- Of course, Qt is well written and [mostly] reliable
- And Open Source, so you can read all of it

- BUT

- There is a LOT of it, and it changes fast.
- It's not practical to review it all.
- Same goes for Linux!!



Safe QML

- Not possible to create certifiable QML for the QtQuick runtime
  - Your QtQuick UI is probably safe, it just can't be certified as safe
- What can you do??!!??
- *The Qt Co is here for you*
- We have a safe renderer

When would you need a safe renderer?

- When you need a system that will not fail?
- Automotive
- Medical
- Railways?
- Aviation & Space

- The Qt Safe Renderer comes pre-certified

- When you need a system that is **certified** as safe
- Certification is about risk management

- Using the right components makes it easier for a certifying authority to verify that appropriate risk management is in place

Qt Safe Renderer

- ISO 26262:2018-6; ASIL D
  - Road vehicles — Functional safety — Part 6: Product development at the software level
- ISO 26262:2018-8 section 11; ASIL D
  - Road vehicles — Functional safety — Part 8: Supporting processes - Chapter 11: Confidence in the use of software tools
- IEC 61508:2010-3 – 7.4.4; SIL 3 and IEC 61508-3
  - Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 3: Software requirements – and Requirements for support tools – 7.4.4
- EN 50128:2011 6.7.4; SIL 4
  - Railway applications – Communication, signaling and processing systems – Software for railway control and protection systems; Software-Tools
- [IEC 62304:2006 C.7, fit- for-use]
  - Medical device software – Software life cycle processes [relationship to IEC 61508 - best practice]. Up to Class C application

OK, enough with the why, let's look at the how.

- Qt Safe Renderer 1.1 offers three safe QML elements
  - SafeImage
  - SafePitcure
  - SafeText
- There is also a SafeMessage element
- You can add these to any standard Qt UI and use them the same as any other QML Element
- Works with Qt Design Studio \o/

The Safe Elements

- Safe Image: takes an image, exactly like an Image but safer.
- Safe Picture: takes a *.dat file, which is a QPIC format. You then add a color overlay onto it.
- Safe Text is Text, but rendered in a safe manner
- Safe Message sends a message to the Safe Renderer runtime from QML

Safe Requirements

- Safe Elements need to be verified as complete, reliable, and visible
  - Size information must be there
  - Elements must be positioned inside the UI boundaries
    - This means the UI boundaries must also be known.
  - Image source files must be set
  - objectName not set will trigger a warning

- There should be no surprises when working with FuSa code, it needs to be predictable

The Qt Safe Paradigm & Process

- You can use the QML elements to create a safe layer that is mapped to the main UI
  - So you would have decorative/cool elements running in the main UI
  - Running over the top: the safe layer
- QSR tools create a series of safe files, that can then be loaded in an entirely separate application
- This gives you two applications:
  - Application 1: Main Application, QtQuick runtime, can not be certified, doesn't need to be
  - Application 2: Safe Application, QSR runtime, can be certified, runs in top layer in separate space
- Application 1 is optional: you don't need to build the Safe app as closely to the Main app
  - It's also possible to monitor the UI in other ways and change the Safe Layout. Approaches differ, QSR is flexible
- NB: QSR is a pre-certified runtime. Using it does not certify **your** code, or guarantee certification!

Controlling the Safe Application

- For a functionally safe UI to be functionally safe, the application can not depend on non-functionally safe systems

- This means the data source *also* needs to be safe
  - You can't send information from the Main Application to the Safe Application

- The QSR API also provides a way to send signals to the Safe Application from C++
  - Messages travel via FuSa channels in your functionally safe operating system

The C++ API

QSafeLayoutResourceReader*

QSafeEvents*

StateManager

The API is not huge

```
enum EventId {
    EventUndefined = 0U,
    EventSetVisibility,
    EventSetPosition,
    EventHeartbeatUpdate,
    EventHeartbeatTimeout,
    EventConnectEventToState,
    EventChangeLayout,
    EventSystemStateChange,
    EventSetText,
    EventSetTextColor
};
```

*Contains no SOUP: the letter Q does denote Qt code inside

The C++ API

- The QSR C++ API compiles into a statically linkable library
- We ship a Safe RCC tool
- You end up with a single binary application to upload

Animations in the Qt Safe Renderer

- Qt Safe Renderer 1.1 series allows you to change properties
- No transition between properties: they just change
- Qt Safe Renderer 1.2 adds in an animation framework!!!

Safe Animations

- QSR 1.2 does offer some animations, but this is not going to completely replace your Qt UI
  - Unless your UX team actually want something pretty simple
- There are limits on what the animations can be used to do:
  - Safe Rendering means no surprises. Everything needs to be cautiously checked beforehand
  - Animations are predefined ahead of time and checked before loading
  - Some animations are actually created by the QSR layout tool!
  - Maximum duration for any animation is 1 second. Longer animations are rejected by the layout tool
- That being said, the QSR will render the animations a up to 64 fps
  - Who says that safety critical software can't be beautiful as well?

How Safe Animations are Created

- Step 1: create a series of States for each of the animatable Safe Elements
- Step 2: add PropertyChanges to the States
- Step 3: define Transitions between these states
- Step 4: add Animations to the Transitions
- Step 5: from the C++, send an event to change the State -->

```
enum EventId {
    EventUndefined = 0U,
    EventSetVisibility,
    EventSetPosition,
    EventHeartbeatUpdate,
    EventHeartbeatTimeout,
    EventConnectEventToState,
    EventChangeLayout,
    EventSystemStateChange,
    EventSetText,
    EventSetColor,
    EventChangeState
};
```

Source Code: QML

› See more in on-demand recording

1.2 Animations: what works, what doesn't?

Yes:

- NumberAnimations
- ScaleAnimations
- Easing types!!!
- Timelines
- Animations on
  - x
  - Y
  - Opacity
  - Width
  - Height
  - Scale

No:

- Animations longer than one second
- Animations that move elements out of bounds
- Animations that are on properties other than x, y, opacity, width, height, scale
- Extraordinarily dynamic safety systems

C++ Sample Code

› See more in on-demand recording

Sample Application!!

› You probably can not run this application at home :-(

In safewindow_p.cpp:

```
SafeWindowPrivate::SafeWindowPrivate(const QSafeSize &size, QWindow *parent)
    ...
//    QWindow::setFlags(Qt::Window | Qt::FramelessWindowHint);
    QWindow::setFlags(Qt::Widget | Qt::FramelessWindowHint | Qt::WindowTransparentForInput);
```

And add the QWindow *parent argument in a few other places so that I can use the safe renderer as a regular QWidget. Of course, you would never do that in production.

Demo Source code from:
https://github.com/reworled/dynamic_telltales
https://github.com/reworled/dat_tool
Non QSR branch is master, QSR branch is qtsaferenderer

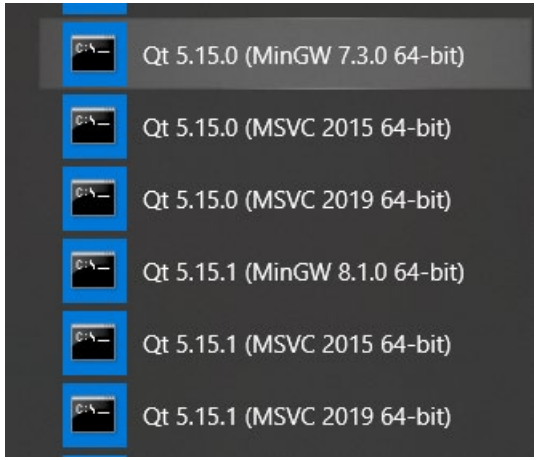
Be warned – this is QSR as a Desktop demo, it is not how you write a certifiable layer!! Demo only!!

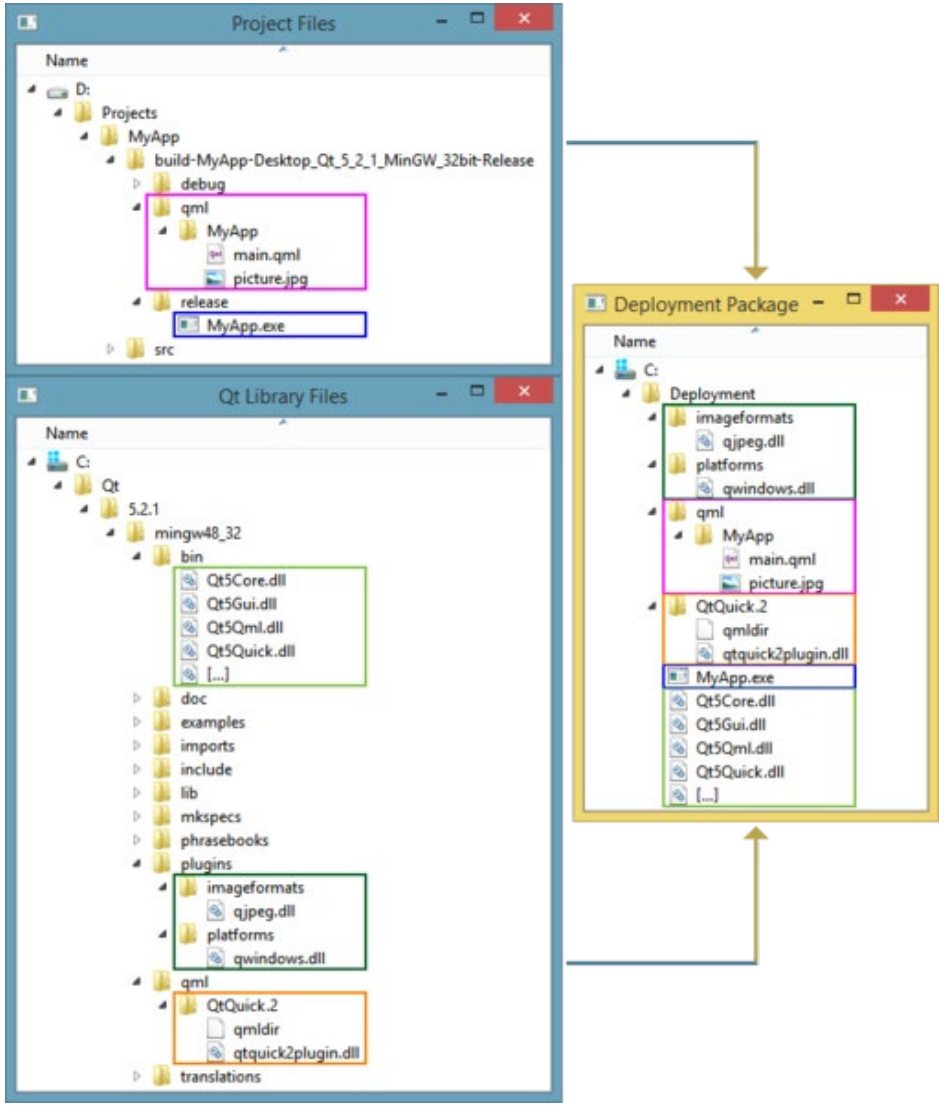
Stay Safe
Render Safe

rebecca.worledge@qt.io

Preparing to deploy applications (Windows)

- **windeployqt**
  - Designed to automate the process of creating a deployable folder
  - Located in QTDIR/bin/windeployqt
  - Needs to be run within the build environment
  - Use script QTDIR/bin/qtenv2.bat (or use pre-installed Qt command line shortcuts)



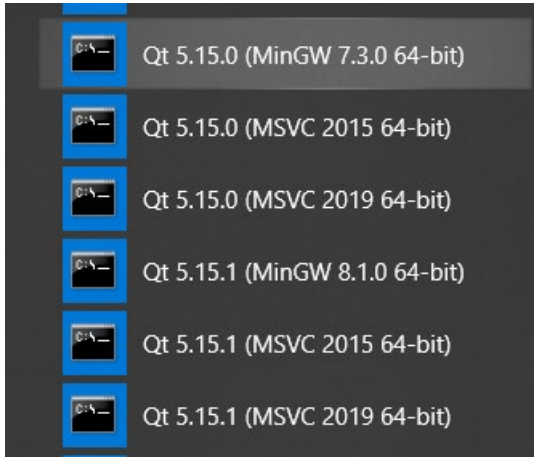


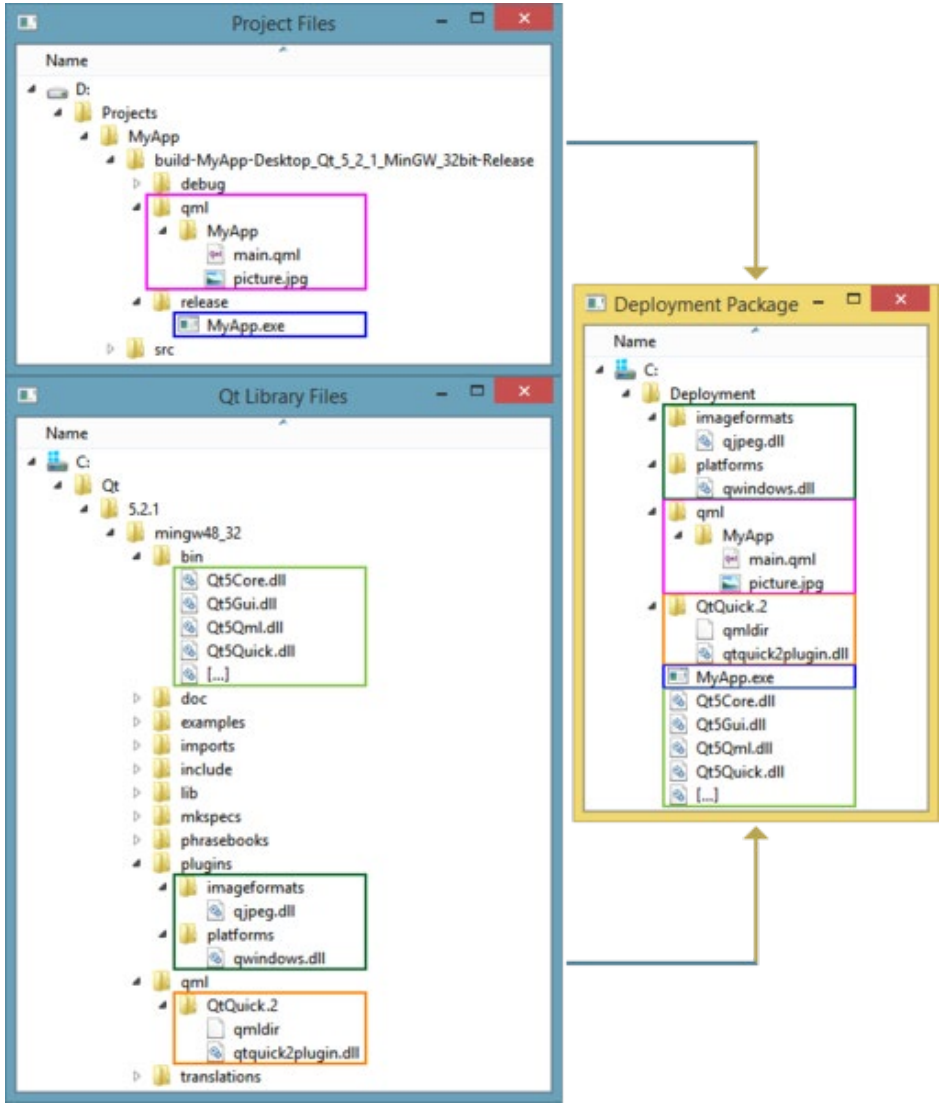
Top 5 Questions

November 2020

Preparing to deploy applications (Windows)

- **windeployqt**
  - Designed to automate the process of creating a deployable folder
  - Located in QTDIR/bin/windeployqt
  - Needs to be run within the build environment
  - Use script QTDIR/bin/qtenv2.bat (or use pre-installed Qt command line shortcuts)





Switching between Qt versions on Ubuntu

- Look for `qtchooser`
  - `-list-versions`
  - `-print-env`
  - `-run-tool=tool`

- Configure
  - *etc/xdg/qtchooser/*.conf*
    System-wide configuration files.
    Each has two lines, the first is the path to the binaries and the second is the path to the Qt libraries.
    If a *default.conf* is provided, the settings from it will be automatically used in case nothing else is selected.
    $HOME/.config/qtchooser/*.conf
    User configuration files.

Light-weight QObject to be exposed to QML

- Do not want to inherit from Qobject

- However, need some reflection capabilities offered by Meta Object System

- Q_GADGET
  - Can have Q_ENUM, Q_PROPERTY, Q_INVOKABLE
  - Cannot have signals or slots

Use Qt Lite with Yocto

- Qt Config tool has now ability to export set of selected Qt Lite features

- How to?

1. Run ./configure for a qt5 source tree with the necessary options for your platform.
2. Open Qt Configuration Tool and select the build directory from the previous step.
3. Select/unselect the features using the tool.
4. When the configuration is ready, select "File" > "Export Features for Boot2Qt" and pick a folder. This will create a feature file for the opted-in modules.
5. In Yocto build, for each opted-in module:
    a. Create a new recipe extension for the module, e.g. "qtdeclarative_git.bbappend".
    b. Add the line "inherit qt5-features" into that bbappend file.
    c. Put the feature file for the module (e.g. "qtdeclarative.opt") into a features/directory next to your bbappend file.

Creating QBSPs from Yocto builds

- QBSP combines toolchain, target device images and set of Qt Creator configurations for a particular device

- For target devices currently supported in the meta-boot2qt layer

```
bitbake meta-b2qt-embedded-qbsp
```

- Target device has not been integrated, what then?
  - Yocto integration is implemented in two classes.
  - Inherit the classes and setup up the required variables

```
qbsp-image.bbclass
qbsp.bbclass
```

LINK: https://doc.qt.io/QtForDeviceCreation/qtdc-qbsp.html


Q&As

Thank you!

info@qt.io

www.qt.io/contact-us