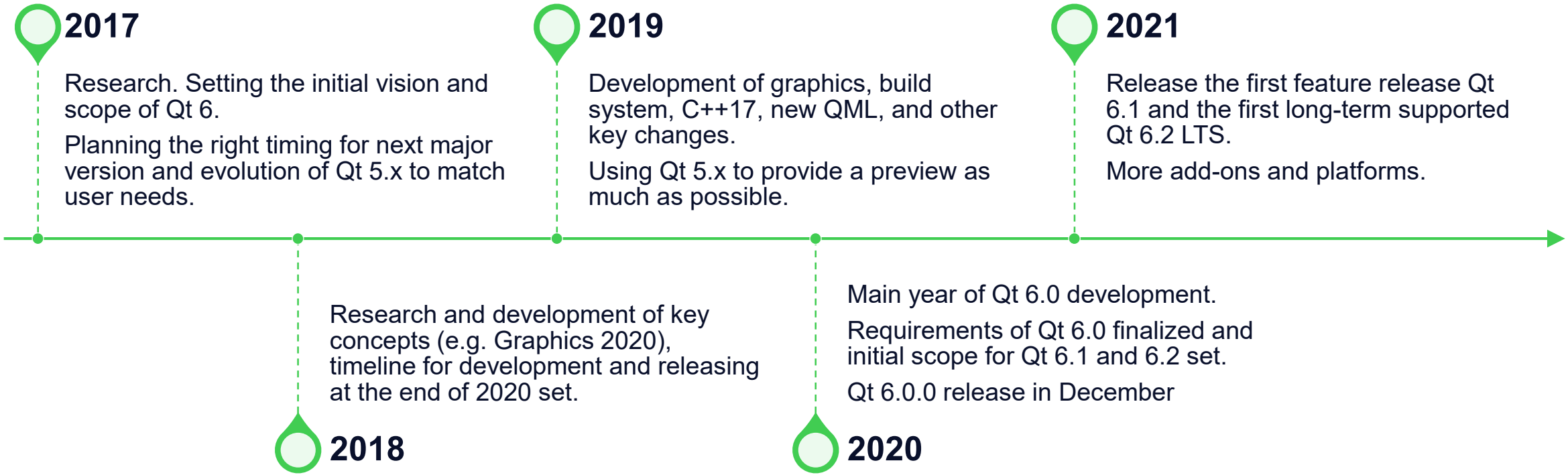
Qt 6.0

Lars Knoll, Qt Chief Architect, The Qt Company
Tuukka Turunen, SVP R&D, The Qt Company

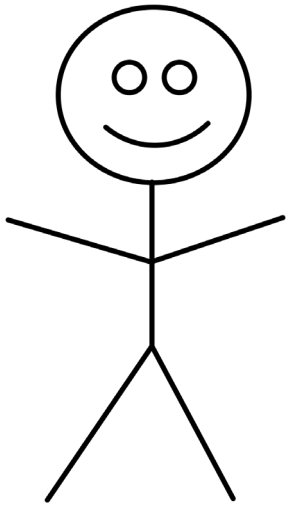
Wednesday 20th January

Qt 6 Timeline

Qt 6 Development Timeline



Package Manager – Additional Libraries for Qt 6

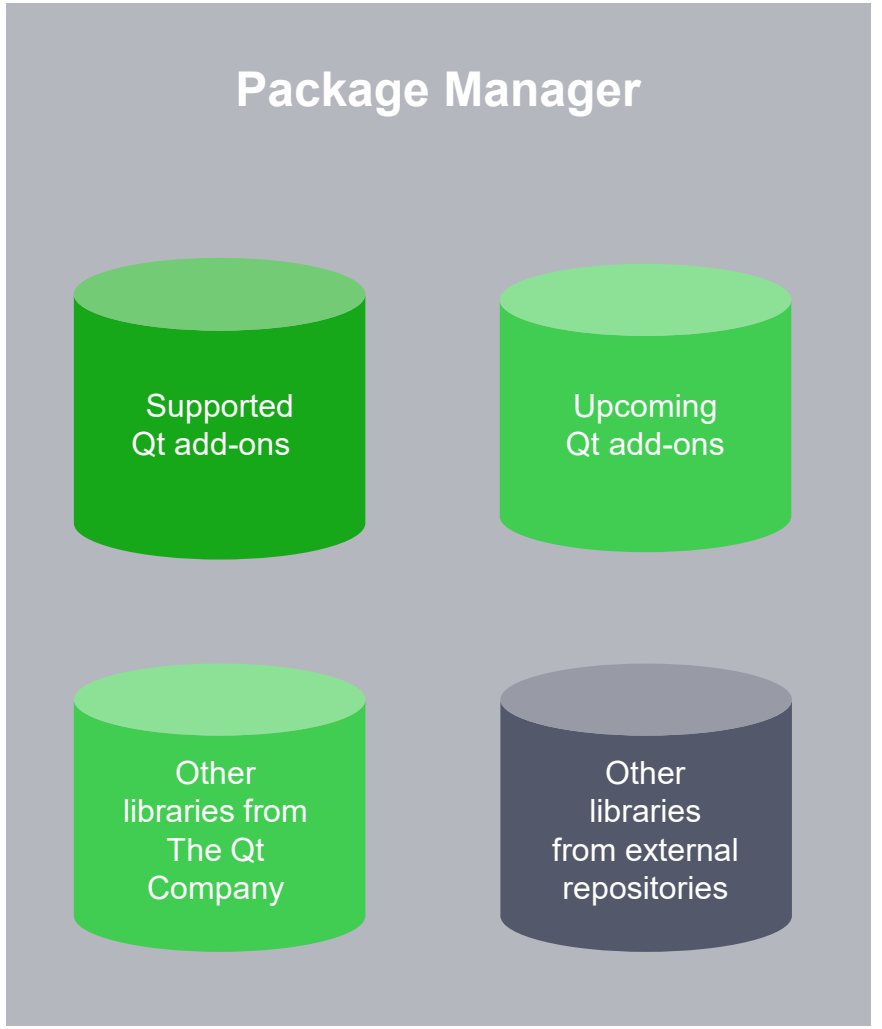


Qt user

Qt 6.x

Qt essentials +
“essential add-ons”

- ✓ Qt Core
- ✓ Qt Gui
- ✓ Qt Network
- ✓ Qt Widgets
- ✓ Qt QML
- ✓ Qt Quick
- ✓ Qt Quick 3D
- ✓ Qt Quick Controls
- ✓ Qt Print Support
- ✓ Qt Wayland + Compositor
- ✓ etc etc



Supported Essential and Add-on Modules in Qt 6

Qt 6.0 (released)

- Qt Concurrent
- Qt Core
- Qt Core Compat
- Qt D-Bus
- Qt GUI
- Qt Help
- Qt Network
- Qt OpenGL
- Qt Print Support
- Qt QML
- Qt Quick
- Qt Quick 3D
- Qt Quick Controls
- Qt Quick Layouts
- Qt Quick Timeline
- Qt Quick Widgets
- Qt Shader Tools
- Qt SQL
- Qt SVG
- Qt Test
- Qt UI Tools
- Qt Wayland
- Qt Wayland Compositor
- Qt Widgets
- Qt XML
- Qt 3D
- Qt Image Formats
- Qt Network Authorization
- M2M package: Qt CoAP
- M2M package: Qt MQTT
- M2M package: Qt OpcUA

Qt 6.1 (planned)

- Active Qt
- Qt Charts
- Qt Data Visualization
- Qt Lottie Animation
- Qt Quick Dialogs (File dialog)
- Qt ScXML
- Qt Virtual Keyboard

Qt 6.2 (planned)

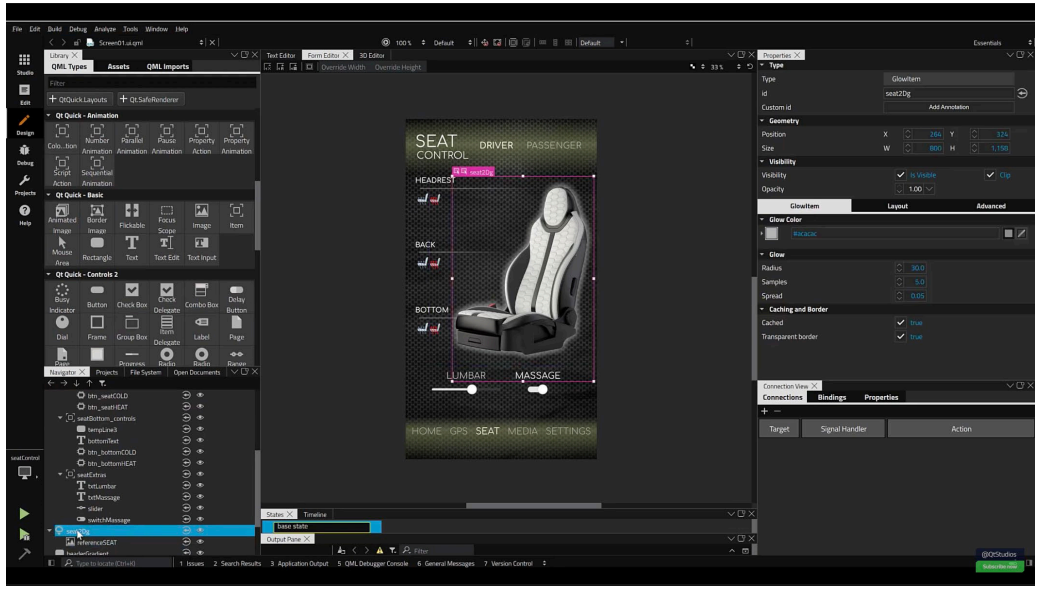
- Qt Bluetooth
- Qt Multimedia
- Qt NFC
- Qt Positioning
- Qt Quick Dialogs: Folder, Message Box
- Qt Remote Objects
- Qt Sensors
- Qt SerialBus
- Qt SerialPort
- Qt WebChannel
- Qt WebEngine
- Qt WebSockets
- Qt WebView

Designer Tools

QT DESIGN STUDIO 2.0

Unified 2D and 3D Design Tool

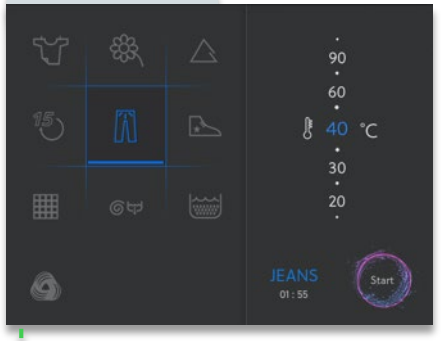
- Create 3D UI's with Qt Quick 3D in addition to traditional 2D UI design
  - Import designs from favorite 2D and 3D tools
  - Visual 3D editor to work with the scenes
  - Define keyframe-based timeline animations
  - Assign 3D effects
- Mix 2D and 3D content seamlessly inside your application



QT DESIGN STUDIO 2.0

Cross Product-line Development

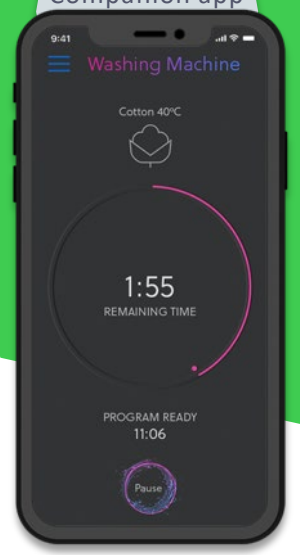
Low-end



Cortex-M4 MCU (<10 EUR BOM) – 640x480

- ✓ Qt for MCUs
- ✓ Smartphone-like UX
- ✓ Basic animations
- ✓ Bare metal or freeRTOS

Companion app



- ✓ Complex/ simple apps
- ✓ Win, Mac, Linux, Android, iOS
- ✓ WEBASM

Mid-range

ARMv7A 32bitlow end MPU (<30 EUR BOM) – 854x480

- ✓ Higher resolution
- ✓ 2.5D Graphics
- ✓ Full Qt Framework
- ✓ Advanced animations
- ✓ Linux or RTOS

High-end

ARM-v8A 64bit Quad Core high end MPU (<100 EUR BOM) – 960x480

- ✓ Highest resolution
- ✓ Dual screen support
- ✓ 2D/3D Graphics
- ✓ Full Qt Framework
- ✓ Linux or RTOS

Developer Tools

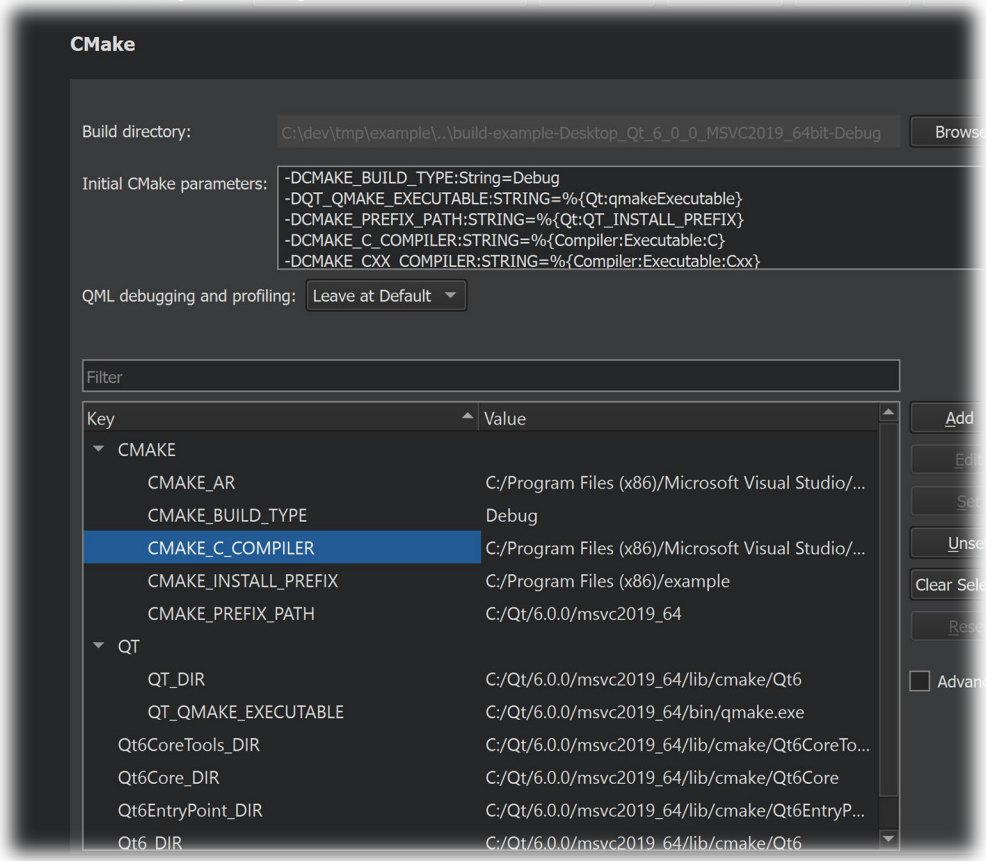
Developer Tooling Fully Supporting Qt 6

- Wizards generate Qt 6 compliant projects
- Inspect Qt 6 types in the debugger
- Editor knows the new QML and C++ language features
- Access Qt 6 documentation and examples

```
cmake_minimum_required(VERSION 3.14)

project(example LANGUAGES CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)

set(CMAKE_AUTOUIC ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(QT NAMES Qt6 Qt5 COMPONENTS Core REQUIRED)
find_package(Qt${QT_VERSION_MAJOR} COMPONENTS Core REQUIRED)

add_executable(example
  main.cpp
)
target_link_libraries(example Qt${QT_VERSION_MAJOR}::Core)

```

Constantly Improving CMake Support

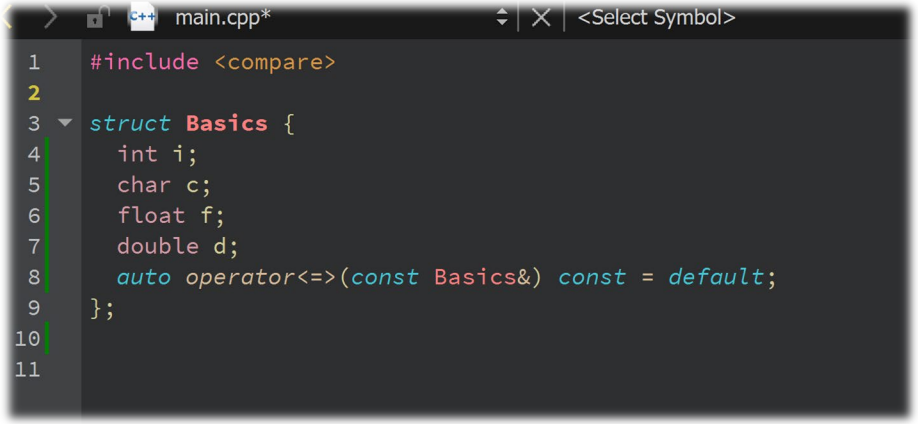
- Improved kit detection when importing builds
- Allow to re-use existing build directory
- Out-of-the box support for QML debugging and profiling
- Applications can still use qmake, if desired

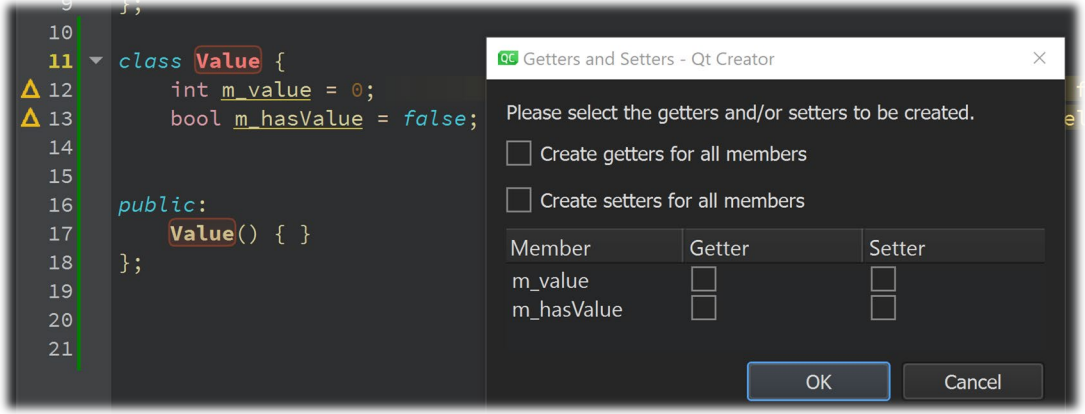


QT CREATOR 4.14

New Tools for C++ Development

- New and improved refactoring operations
- Clang code model to Clang 11
- Tighter integration of Clazy and Clang-Tidy
- Multiple improvements to C++ development throughout





Qt 6

Goals

> Fix architectural limitations in Qt 5
> Improved performance
> House cleaning
> Package management support
> Smaller and modular core product
> Compatibility with Qt 5

Core Values

> Cross platform
> Maintainability and compatibility
> Scalable
> Intuitive and easy to use
> Documentation
> Tooling

Focus on cleaning up our code base and architecture, not on new features

Compatibility

> Mostly source compatible with Qt 5
> Some porting required

- Requires some changes to source code
- Requires a recompile
- Deprecated functionality has been removed

> Those changes prepare us for the years to come

What’s new in 6.0?

Require C++17

- Require a C++17 compliant compiler
- Use features available in C++17
  - Structured bindings
  - if constexpr ()
  - Template argument deduction rules
  - New std library features
  - …

Containers

> Closed API gaps towards STL containers

- Consistent support for move operations
- Emplace
- initializer_list

> Removed 2G size limitation

- qsizetype

> Default constructors constexpr and non allocating

QList and QVector

> Only one class

- QVector is an alias to QList

> Simplifies our API

> Removed performance issues in Qt5 QList

- vector-based data structure
- Improves performance in almost all cases

> Almost 100% source compatible

- Exception: References to stored items not stable under modification

QMap and QHash

> Separated QHash/QMap and QMultiHash/QMultiMap

- Don’t inherit from each other anymore

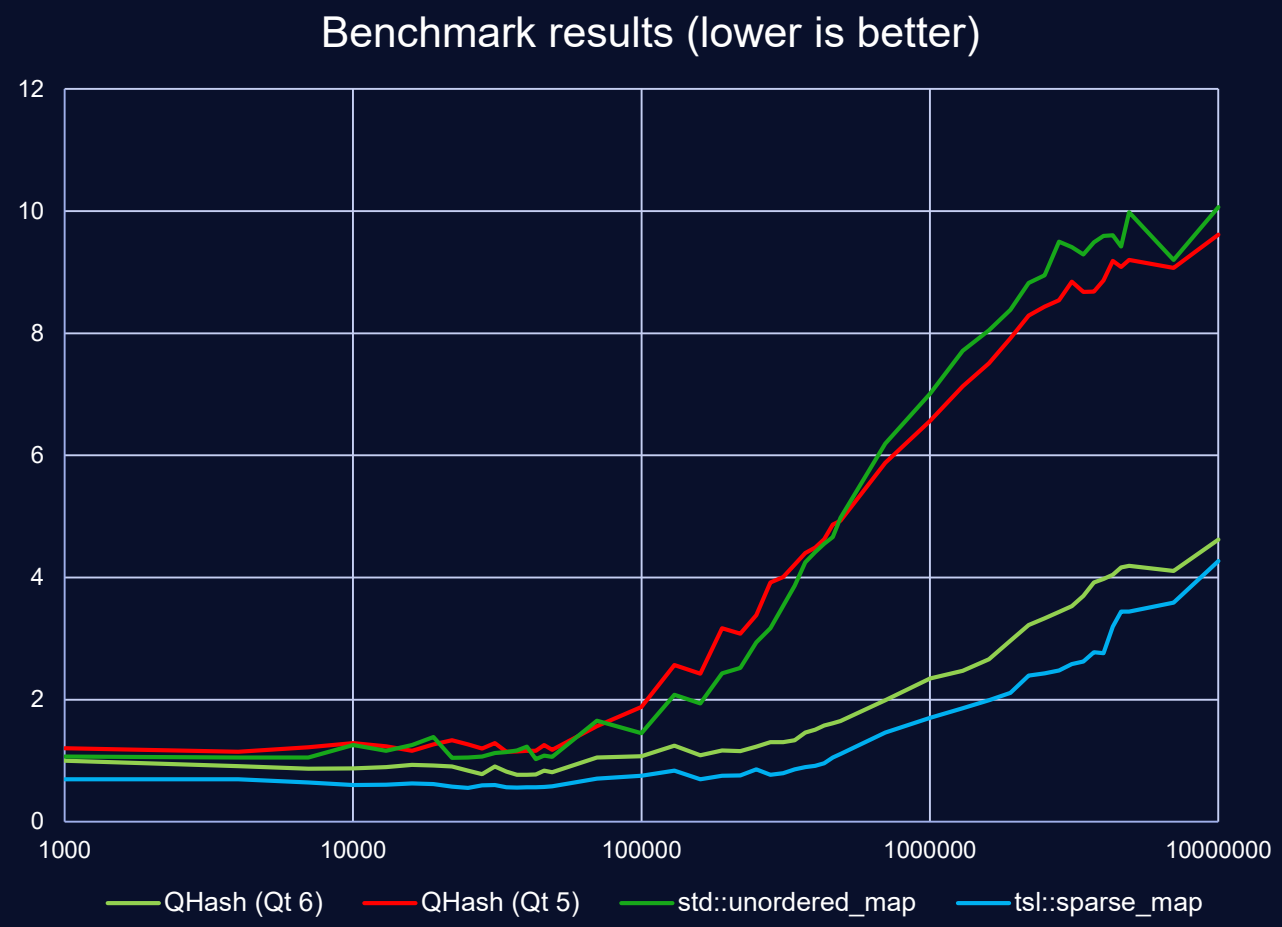
> Q(Multi)Map implemented on top of std::(multi_)map

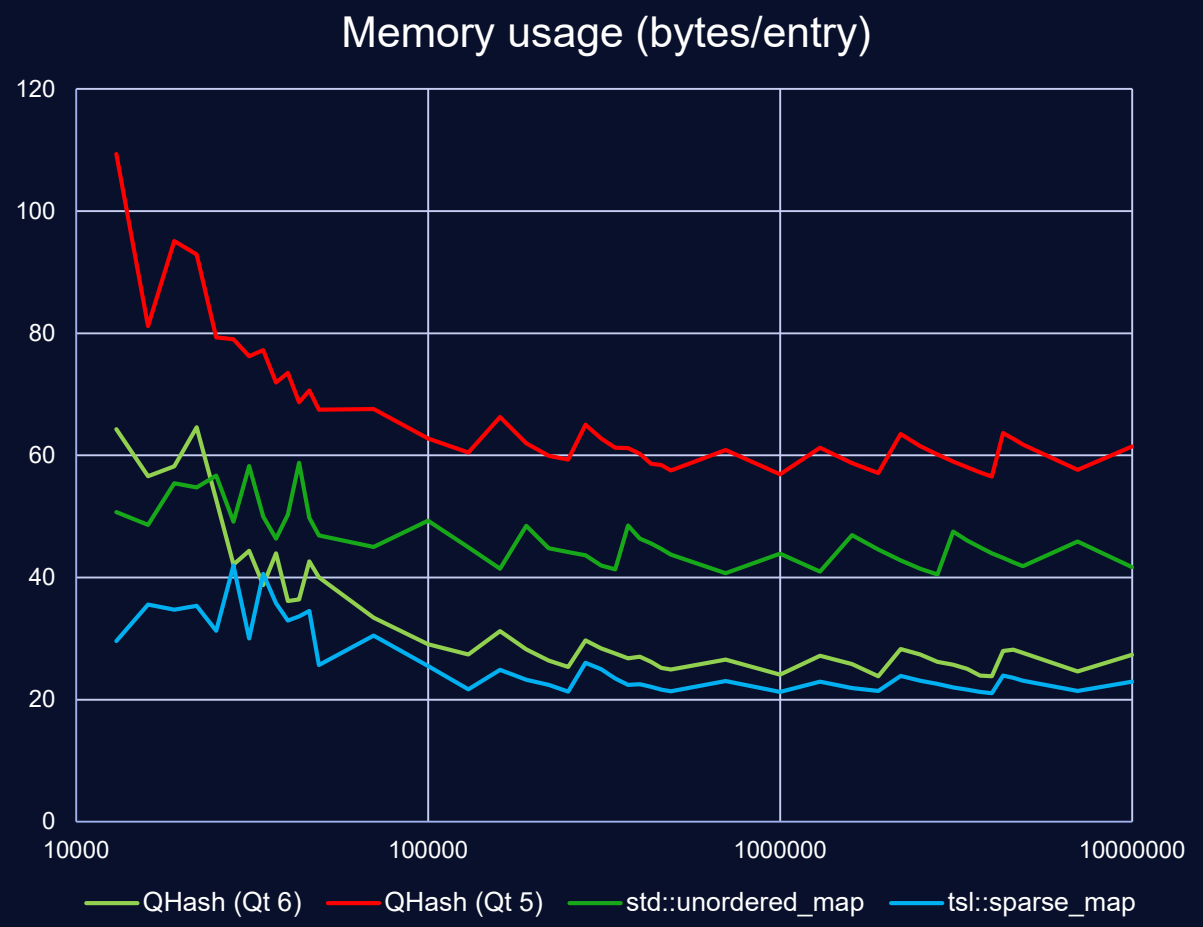
- Move support from/to std::map
- Implicitly shared

> New QHash implementation

- Open Adressing
- Fast
- Low memory usage

Performance: Hash<int64_t, int64_t>





Measurements done using
https://github.com/Tessil/hash-table-shootout.git

Unicode & String classes

> All text handling Unicode based

- UTF-16 the default string encoding
- UTF-8 the default storage format

> Source code is UTF-8

> Clean up string related classes

- Consistent set
- QUtf8StringView

> Move legacy encoding support out of Qt Core

String classes

> **QByteArray**
  - ASCII only, not Latin1
  - Meant for storing binary data

> **QUtf8StringView class**
  - Wrapper to tag UTF-8 specific data

> **Completed QStringView API**
  - Supports the const methods of QString

> **Removed QStringRef**
  - Still available in Qt5 compatibility library

Regular expressions

> Qt 6 uses QRegularExpression everywhere

- Only one regular expression engine
- Based on PCRE2

> QRegExp has been removed

- Porting to QRegularExpression usually straightforward
- Still available in Qt5 compatibility library

Text conversions

> New QStringEncoder/Decoder API

- Value based, stateful
- Supports UTF-8, UTF-16, UTF32, Latin1 and Local 8 bit
- Extensible to support full code conversions after Qt 6.0

> QTextCodec has been removed

- Still available in Qt5 compatibility library

Properties and bindings

```
import QtQuick 2.15

Rectangle {
    property int width
    property int height: width
    property int border: height/10
}
```

Goals

> Bring the best part of QML to all of Qt

> Move binding infrastructure into Qt Core

- Extend existing properties in QObjects with support for bindings
- Add binding support to any other class

> Support lazy binding evaluation

- Mark bindings as dirty when dependency changes
- Only re-evaluate when value is required

> Make it fast

```
struct Rectangle {
    QProperty<int> width;
    QProperty<int> height;
    QProperty<int> border;

    Rectangle() {
        height.setBinding(Qt::makePropertyBinding(width));
        border.setBinding([this]() {
            return this->height / 10;
        }
    }
};
```

```
template <typename T> class QProperty {
    std::function<T()> binding = nullptr;
    T data;
public:
    T value() const {
        if (binding) return binding();
        return data;
    }
    void setValue(const T &newValue) {
        if (binding) binding = nullptr;
        data = newValue;
    }
    void setBinding(std::function<T> b) { binding = b; }
};
```

```
template <typename T>
class QProperty {
    T val;
    QPropertyBindingData d;
public:
    T value() const {
        if (d.hasBinding()) d.evaluateIfDirty(this);
        d.registerWithCurrenlyEvaluatingBinding();
        return this->val;
    }
    void setValue(const T &t) {
        d.removeBinding();
        if (this->val == t) return;
        this->val = t;
        notify();
    }
};
```

QObject complicates the picture

> Existing property infrastructure

- Avoid breaking compatibility with Qt 5
- Extend the existing property()/setProperty() mechanism

> Data hiding

- Data lives in QObjectPrivate

> Avoid overhead if bindings aren’t used

Solution

> Binding support a simple addition to the existing system

> Add a getter for a binding interface

- QBindable<PropertyType> bindableProperty();

> Extend Q_PROPERTY with a BINDABLE tag

- Tells moc about the binding interface

> Use Q_OBJECT_BINDABLE_PROPERTY to implement the data storage

- Only works as a member of a QObject

> Low overhead if bindings aren't used

- One additional pointer in QObjectPrivate

```
class MyObject : public QObject {
    Q_PROPERTY(int x GET x SET setX BINDABLE bindableX)
    Q_OBJECT_BINDABLE_PROPERTY(MyObject, int, xData)
public:
    int x() { return xData; }
    void setX(int x) { xData = x; }
    QBindable<int> bindableX() { return &xData; }
};


myObject->bindableX().setBinding([otherObject]() {
    return otherObject->x() + otherObject->width();
}
```

QMetatype and QVariant

> Common backend for QMetaType and QVariant

- QMetaType can finally handle all types

> High performance, fast code paths

- QMetaType contains one pointer to a handler struct
- Automatic registration
- Automatic support for comparisons and QDataStream

> Deprecated integer-based type id

- Use QMetaType directly

```
struct QMetaTypeInterface {
    uint size, alignment;
    const char *name;
    ConstructFn construct;
    DestructFn destruct;
    EqualsFn equals;
    ...
};

class QMetaType {
    QMetaTypeInterface *iface;
public:
    const char *name() const { return iface->name; }
    void construct(void *where) { iface->construct(where); }
};
```

Improvements to moc

> Store and resolve required metatype information at compile time

- No more string lookups for type information
- Faster and more efficient runtime resolution of methods

> Added support for bindable properties

- Dynamically add and remove bindings

Rewritten Concurrent

> Simple chaining of computations
  - `QFuture::then()`
> Attach failure and cancellation handlers
  - `QFuture::connect()`
> Convert signals to QFuture objects
> Added a QPromise class
> Support for custom thread pools

```
QByteArray download(const QUrl &url);
QImage loadImage(const QByteArray &data);
void show(const QImage &image);

auto future = QtConcurrent::run(download, url)
                    .then(loadImage)
                    .then(show)
                    .onFailed([](QNetworkReply::NetworkError) {
                        // handle network errors
                    });
```

Further improvements

> Multiple improvements in Qt Network

- SSL/TLS improvements
- HTTP/2 support by default
- Write your own plugins for QNetworkAccesManager

> Added Qt5Compat module

- Ease porting to Qt 6
- Contains some deprecated classes from Qt 5

Unified Pointer events

> Unified handling of mouse, touch and tablet events

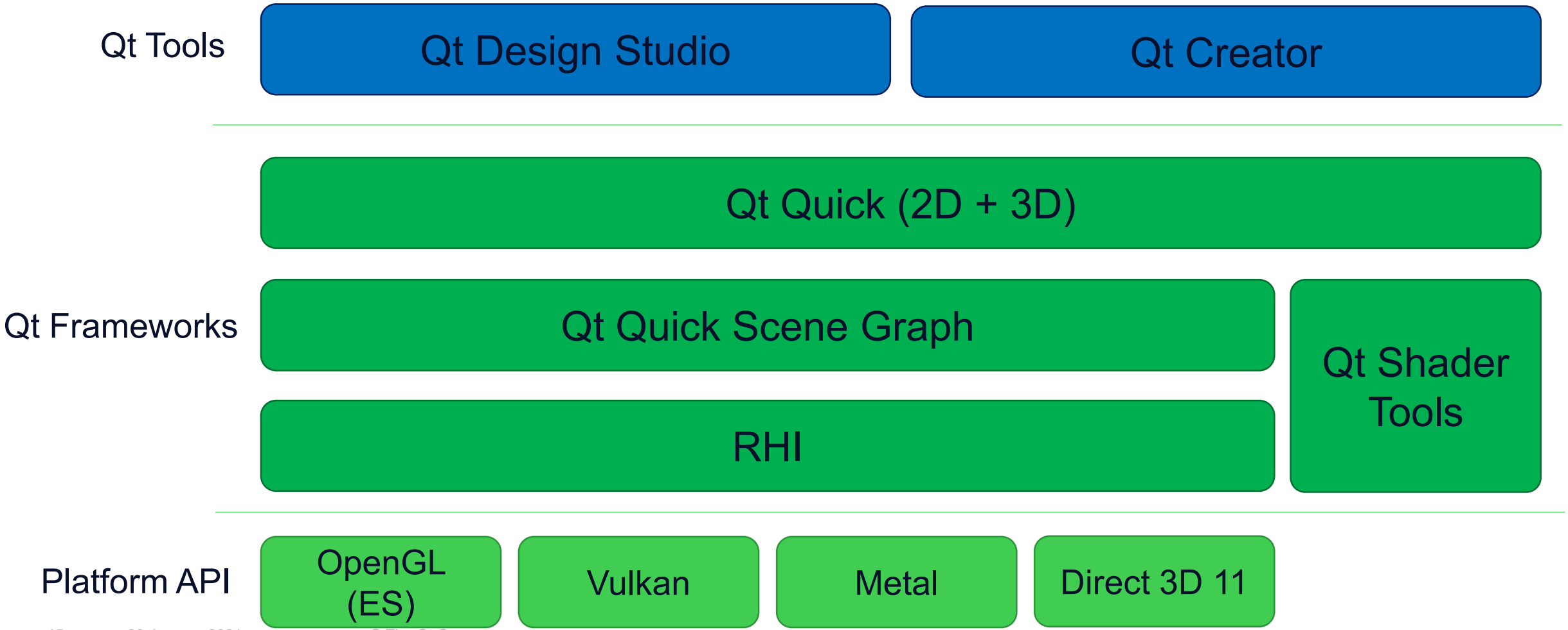
- One common base class: QPointerEvent
- Specializations for mouse, touch and tablet events
- Common data
- Tracking of input device and history of event points

> Resolves recurring issues with touch handling in complex controls

- e.g. touch enabled controls inside Flickable

Graphics

Graphics Architecture



Rendering Hardware Interface

> Abstraction layer for 3D graphics APIs

- OpenGL, Metal, Direct 3D 11, Vulkan

> Abstracts graphical objects

- Materials, Meshes, Shaders, etc.
- Internal API tuned towards the needs of Qt

> Integrated with Qt Platform and Window system abstraction layers

> Qt Quick and Qt Quick 3D are fully ported to RHI

Qt Shader Tools

> Support for cross platform shaders

- Write shader once
- Recompile to all graphics APIs

> Support for build time and runtime shader compilation

- Dynamically generate shaders at runtime
- Bake the shaders offline

Qt 6 uses the native graphics API of each operating system

Quick 3D

> Efficiently combine 2D and 3D in one scene

- One optimized scene graph

> Enhanced PBR support

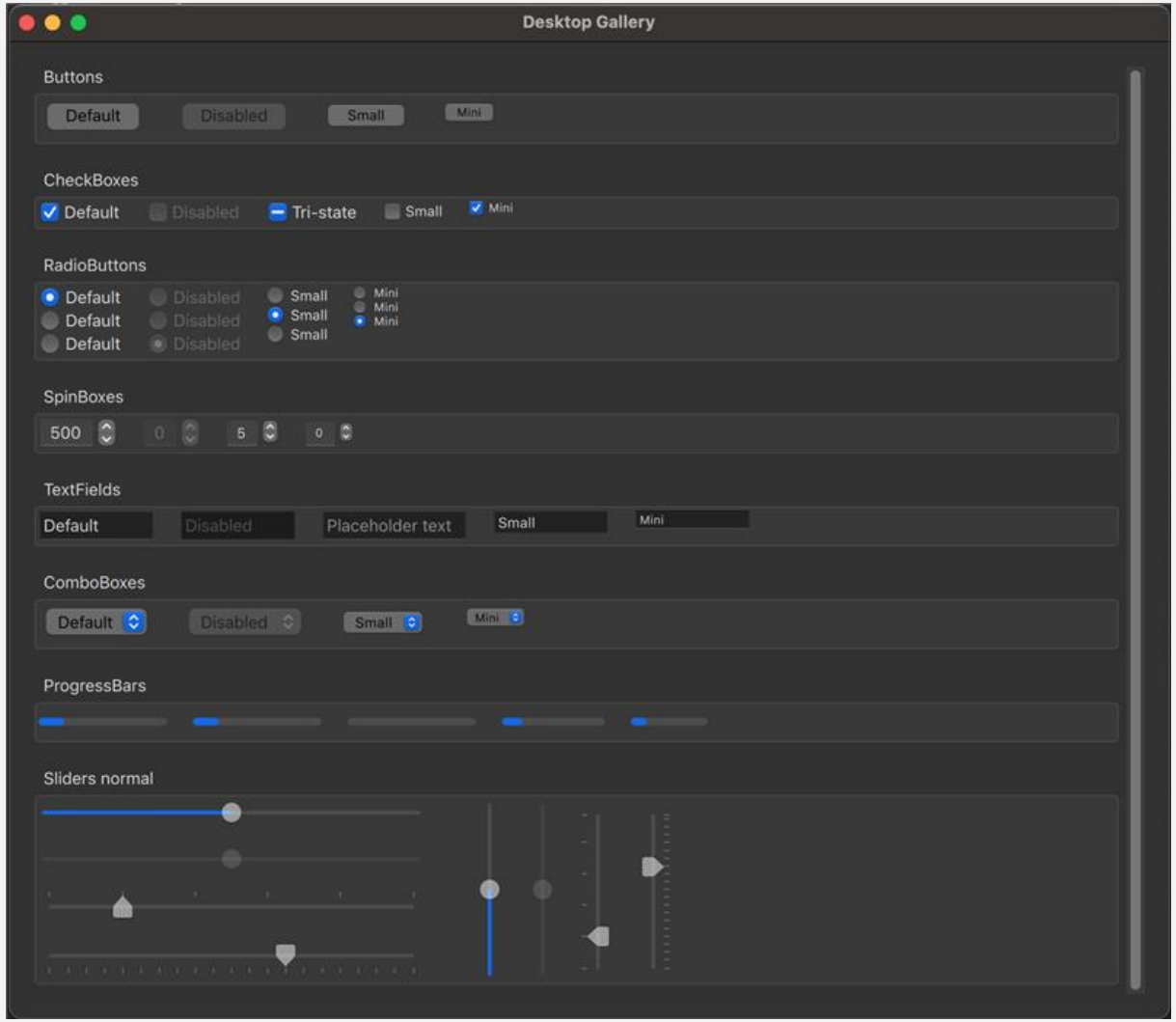
- Materials look like they are designed

> Greatly improved support for GITF2

- Support base spec
- Most of the common extensions

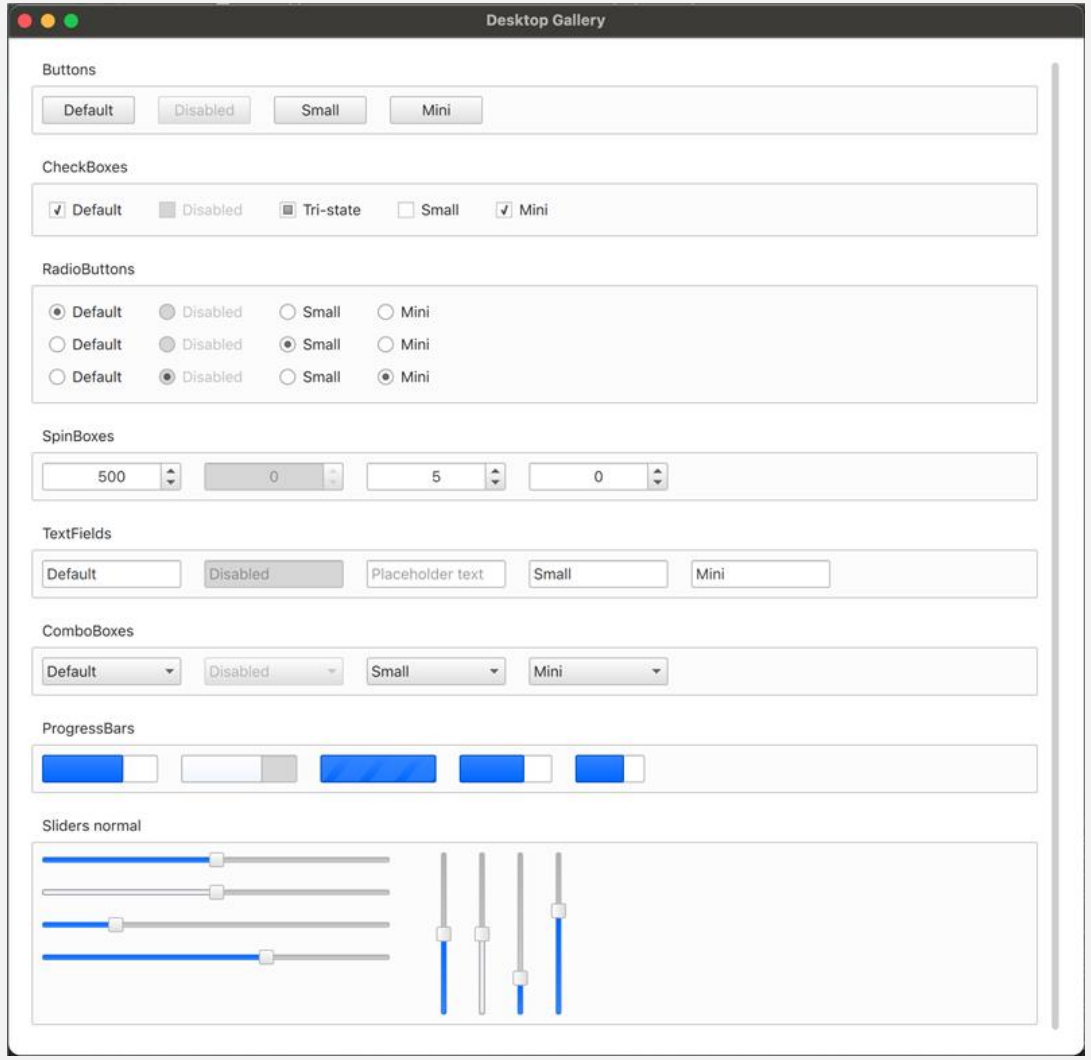
Desktop style for Quick Controls

Native look and feel for Qt Quick Controls on macOS, Windows and Linux

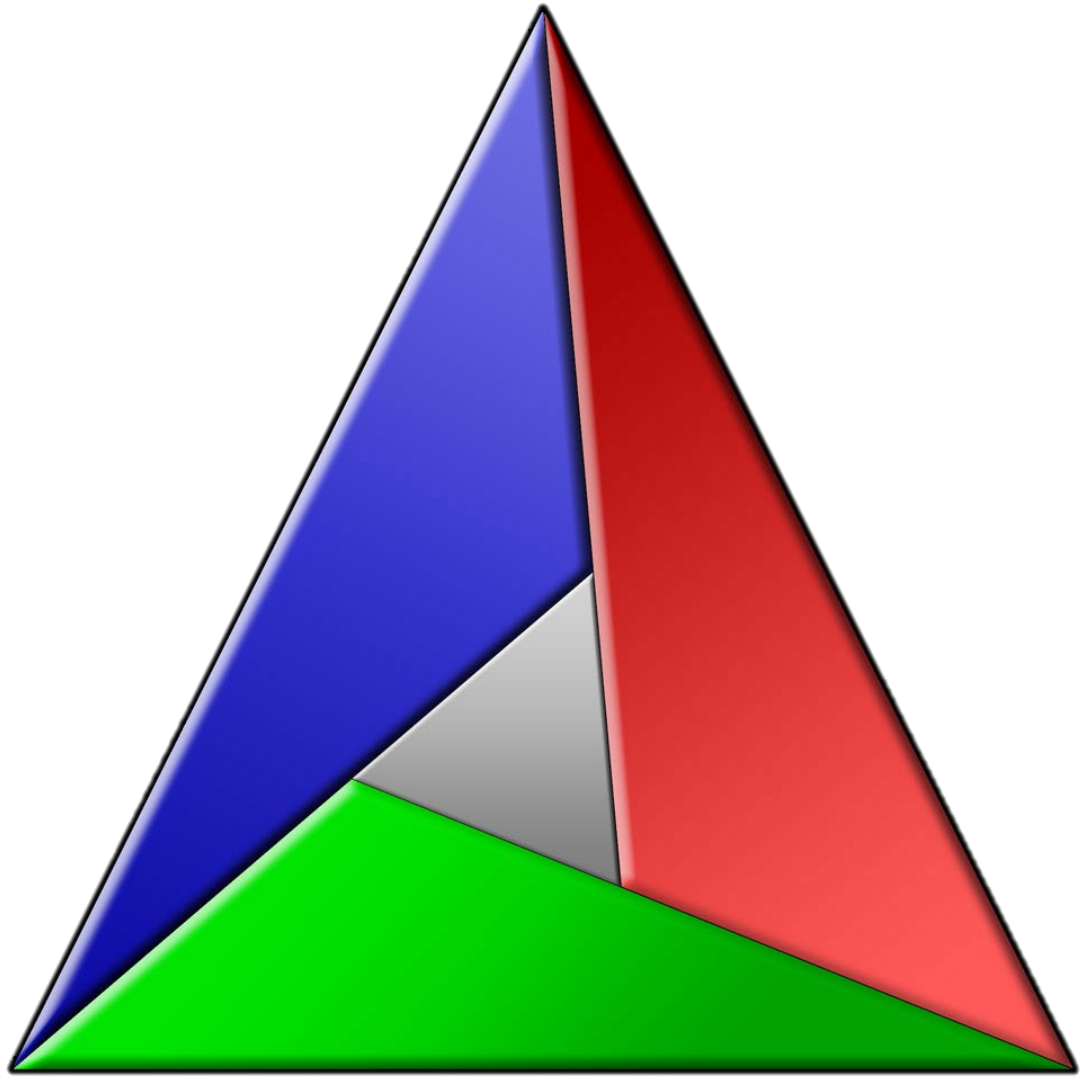


Desktop style for Quick Controls

Native look and feel for Qt Quick Controls on macOS, Windows and Linux



CMake



Supported platforms

> Host platforms

- Windows 10
- macOS 10.15 and 11.0
- Linux (Ubuntu 20.04, CentOS 8.1, SLES 15, OpenSUSE 15.1)

> Target platforms

- All host platforms
- Yocto 3.1 Dunfell
- iOS 13/14
- Android (28 built-time, 21 run-time)

Roadmap for other platforms

> macOS on ARM

- Timing not yet confirmed

> Yocto 3.2

- Planned for Qt 6.1

> QNX & INTEGRITY

- Technology Preview with Qt 6.1
- Full support planned for Qt 6.2

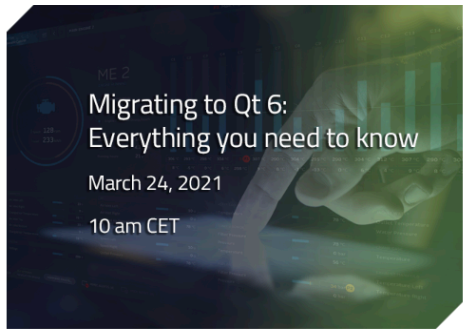
Release roadmap

- **Qt 6.0** December 2020
- **Qt 6.1** End of April 2021
- **Qt 6.2 LTS** End of September 2021

Outlook

- **Porting the remaining Qt Add-ons to Qt 6**
  - Delivered through the package manager as they are ready
- **Make most properties bindable**
- **Wider C++ API for Qt Quick and Qt Quick Controls**
- **Improvements to QML performance**
- **Improvements in native styling, accessibility and graphics features**

Want to Know More?

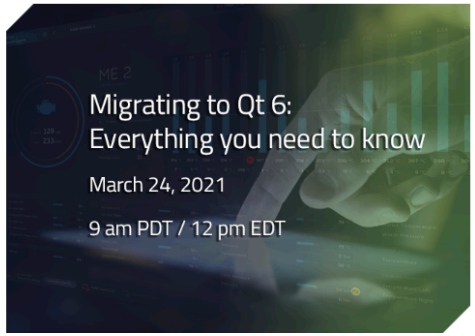


Migrating to Qt 6: Everything you need to know

Online ▪ 10:00 Mar 24, 2021 UTC +1

Register View All Live Webinars ›

Share with your friends



Migrating to Qt 6: Everything you need to know

Online ▪ 9:00 Mar 24, 2021 UTC -7

Register View All Live Webinars ›

Share with your friends

Thank you!

info@qt.io
www.qt.io/contact-us

Lars Knoll, Qt Chief Architect, The Qt Company
Tuukka Turunen, SVP R&D, The Qt Company