EBOOK

Qt Conquers the Automotive Interior

About this eBook

To successfully woo drivers and passengers, the car's interior screens – digital instrument clusters, heads-up displays, infotainment systems, and rear-seat entertainment systems – have always required responsive graphics and reliable performance. Increasingly, these displays must also meet functional safety standards with very rapid design-to-development production cycles.

This eBook is about the development of today's sophisticated digital car interiors and why Qt turns out to be the platform of choice. We'll look at trends driving automotive user interfaces and see how Qt addresses them. We'll delve into the special relationship between UX designers and software engineers to see how that interaction can be streamlined with Qt. And we'll end by examining a couple of case studies where Qt in the car comes together.

The challenges of building for automotive

As every automotive engineer knows, building an automotive system is not just a matter of slapping together mobile-phone and navigation technology and putting it into a dash. An automotive platform must adhere to stringent standards – both internal and external – for quality and reliability. It must be built with the lowest possible cost, operate smoothly at 60 frames per second, and make parts available for fifteen years. It must support multiple different geographies and cultures, and be customizable across model lines. It must also be booting fast to allow the driver to operate with the information provided by the instrument cluster as soon as the ignition key is inserted.

The automotive space is in the process of transforming itself with a whole new set of consumer-driven demands – and yet it still must observe the requirements that have made it reliable and trustworthy to date. This is currently the central problem of modern automotive design: introducing dynamic innovation without also introducing risk.

Setting the stage: The current state of the in-car experience

The evolution of the in-car experience has been dramatic and swift. It wasn't that long ago when the radio was our only conduit to the outside world. Ubiquitous connectivity has changed that and displays throughout the vehicle now dominate:

- The primary screen in today's car is the infotainment system or center stack. These systems originally started as a premium option and have gradually become standard for every car except the most basic entry-level models … for now.
- Digital instrument clusters that replace the analog gauges behind the steering wheel are following the infotainment screen trend. They're currently common in luxury vehicles but are starting to appear in mid-level ones as well.
- Mobile devices were predicted to kill off rear-seat entertainment systems but that hasn't happened. Families find it too convenient to have this always-available entertainment for their children that they can control from the front seat.
- Heads-up displays – instrument data for the driver that appears behind the windshield – are still an uncommon option. Augmented reality in autonomous cars may revive this product category but will likely focus on use cases that aren't driver-exclusive.
- Co-pilot screens aren't in production cars now but have been demonstrated – as well as full dashboard graphical displays. Expect "full glass cockpits" to start appearing with the rise of self-driving cars.

What's happening to the automotive UX – and how is Qt relevant?

The displays in this year's cars were designed and created two or more years ago. Today more than ever before, there is incredible pressure to drastically cut that development cycle so that automotive electronics can stay current with other consumer products. There are several factors impacting automotive UIs, many of which are responsible for directly expediting this product life-cycle truncation.

User experience expectations

The ease of use, beautiful animations, and natural interfaces of our ever-pervasive smart phones have biased our expectations for every device, and this includes in-car displays. Consumers now expect their cars to perform as well as their mobile devices – and nothing less.

Qt is designed to create a great looking UX and smooth animations through QML or Qt Widgets. With its strong cross-platform support, Qt is frequently used for developing mobile applications for iOS and Android devices, which means that those same powerful UX idioms and frameworks are readily brought over to the automotive world.

Large touchscreen displays

As the "wow factor" of car electronics has become a significant factor in purchasing decisions, automakers have been increasingly expanding their infotainment screen size and resolution. They have also become further emboldened in pursuing larger screens in part thanks to the public praise Tesla cars have received on their massive infotainment displays. Another important trend is the touch display, one of the most intuitive and modern input mechanisms.

Qt has all the necessary technology to easily create intuitive touch-driven displays. And while the size of the display is primarily a hardware concern, larger screen sizes and resolutions can dramatically multiply the amount of video and image data being generated and moved, directly impacting software performance. As one of the fastest graphical platforms, this is an area where Qt shines.

Incorporating 3D

Let's face it – 3D graphics are cool looking. This is especially true when it comes to physical-based rendering (PBR) in digital instrument clusters, which lets automakers closely emulate the appearance of physical dials and gauges to which drivers are accustomed. It also gives automakers the freedom of making changes to the design or visuals at any time to create a consistent cockpit hardware platform across vehicle models.

Qt 3D makes manipulating and displaying 3D models a snap. With unique flexibility at the heart of its design, the Qt 3D engine can use basic 3D rendering with materials and meshes, or sophisticated modern rendering techniques like PBR and ambient occlusion, and can be easily extended. This rendering flexibility is not a feature that you'll find in other graphical frameworks.

Personalization

In a world where it's easy to categorize and digitize people's identities, it makes sense that we would want to preserve our uniqueness. The wallpapers, ringtones, and themes we select for our digital devices let a bit of our personalities shine through, and that desire for personalization has made its way into the car.

Qt makes it easy to dynamically reskin a UI, allowing users to select the instrument cluster of their choice or an infotainment background that matches their mood. Needless to say, this ability also comes in very handy for internationalization purposes as automakers supply global markets with common hardware platforms.

Mobile + home integration

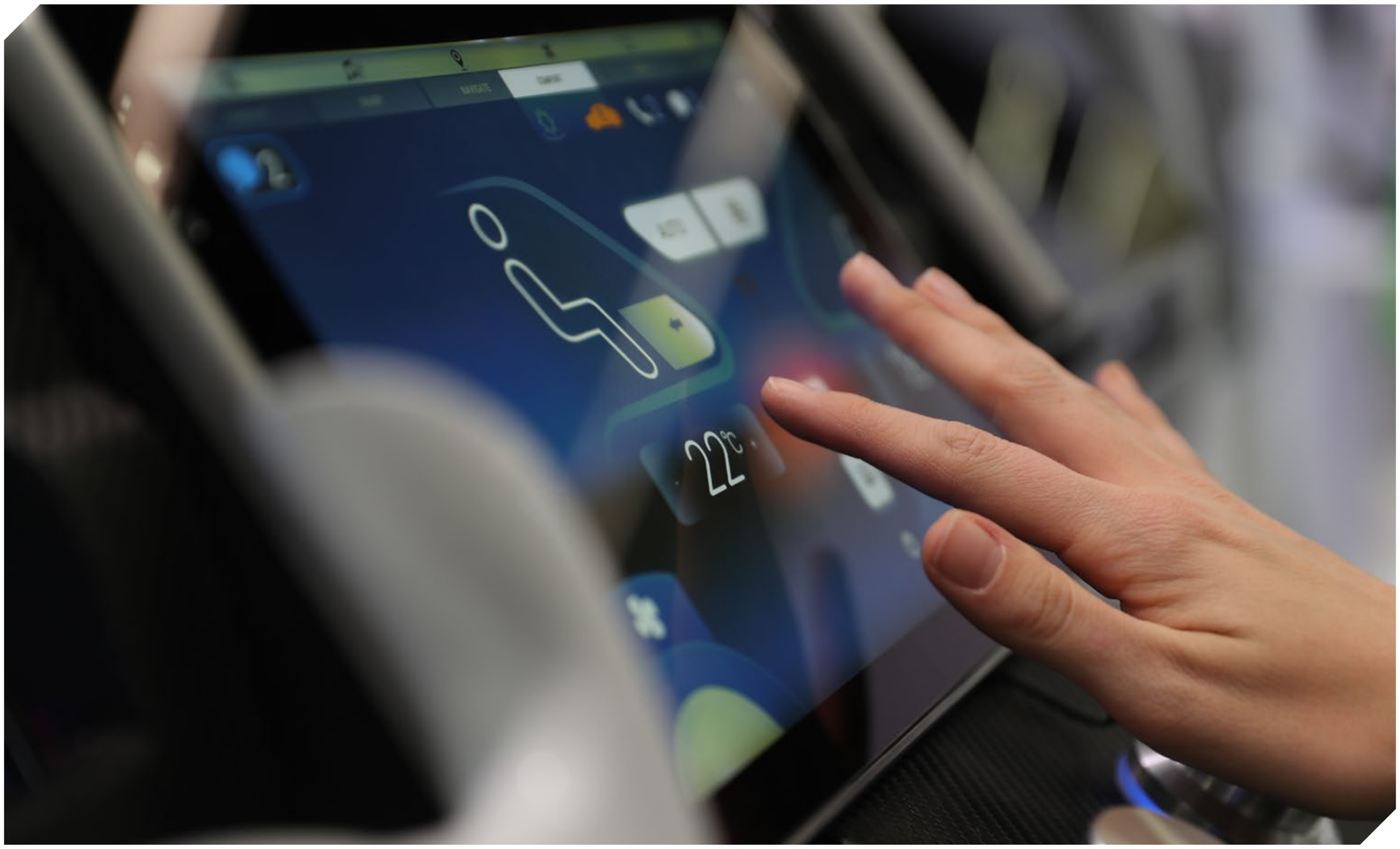
Consumer desire to have phones integrate with cars has caused every automotive OEM to incorporate support for Apple Car Play and Android Auto into their models. And as IoT devices and digital voice assistants begin to enrich our lives and fill our homes, it's only natural that we would want access to their convenience everywhere, including the car.

Qt has support for all the protocols and transports needed to seamlessly interface with mobiles, houses, and IoT devices. With broad support for all of these areas within the Qt ecosystem, Qt components and libraries for the car will undoubtedly be able to keep pace with consumer developments.

Autonomous driving

While the reality of self-driving cars may take a couple years to take hold, significant changes to the automotive interior are on their way. Without concern for driver distraction, the interior can support completely new user experiences, such as full graphical cockpits that visualize the car's intelligence or dynamically shift from gaming console to home theater.

Qt can easily manage multi-screen displays, video playback, Internet browsers, as well as standard infotainment screens and vehicle controls.



Electric vehicles (EV)

Every major OEM has announced new significant goals in electrifying their vehicle fleet. Because internal combustion engines are very complex to manufacture, the move to EV also makes building a car easier, allowing many new entries into the market. As a result, software is becoming more of a differentiator than it already is – the new "automotive arms race" is the length of the development cycle and the power of the software platform. The other new big requirement in interior electronics thanks to EVs is minimal electricity consumption, as every wasted watt decreases the car's distance.

Qt is well suited as a platform for rapid development; something that we'll discuss later while reviewing how OEMs can maintain their differentiation. The battery-sipping nature of EV electronics also naturally leads to using Qt for automotive interior displays. Because Qt and C++ are built to maximize performance and minimize CPU cycles, they can draw less current from the battery to help enable longer driving cars.

Augmented reality (AR) and virtual reality (VR)

AR home mechanics that help laypeople troubleshoot under the hood and virtual field guides for kids on vacation have recently been popping up on the tradeshow circuit. VR design tools used internally by automakers and dealership tools where consumers can explore their dream cars using a VR headset are also starting to gain some traction.

Because of Qt's excellent 3D graphics with support for multiple composition layers, AR and VR applications are possible now. Active players in the Qt community are also exploring adding a more packaged capability to Qt through a Qt 3D VR module.

The new realities in building automotive systems

Don't assume that market trends are only affecting the way cars look; there are significant shifts happening in how cars are designed and built. The rising dominance of software within the car, the car's growing dependency on connectivity and cloud, and an emerging holistic view of the vehicle's software architecture – as outlined in McKinsey's February 2018 report "Rethinking Car Software And Electronics Architecture" – are just a few of the forces resetting assumptions of how we create car software. We outline how the shifting landscape of automotive software can be elegantly addressed with Qt.

✓ Built-to-spec versus standardized platforms

To build vehicles and preserve product differentiation, automakers have typically created huge specifications detailing every feature and function, with third party companies constructing sub-systems from the ground up each time. This model creates tightly coupled hardware and software – modules that are impossible to reuse outside of their original intent. This results in a new specification and new product for every vehicle line, dramatically slowing the speed of innovation.

Today, automotive electronic differentiation isn't in the core platform features – it's in the services offered and speed of reacting to new trends. This has caused OEM product teams to slowly shift to a process that can embrace adding their "secret sauce" to an existing platform. By using modular platforms that are hardware agnostic and adopting agile development methodologies, automakers are free to create branded experiences both inside and outside the car.

Qt Automotive Suite is a comprehensive toolkit built on top of the Qt framework. It provides a stable and highly functional core platform for building infotainment systems, instrument clusters, and rear-seat entertainment, with a readily modifiable reference design, the necessary middleware pieces already in place, and custom tools to design, debug, and deploy.

✓ Time-to-market pressure

Automotive software used to be frozen at vehicle launch and changed only rarely, if ever – a rigor that required software to be as bug-free as possible.

Due to the rapid release cycles of weeks or even days between updates on consumer electronics and mobile apps, customers have become used to a rapid cadence of features – and fixes to problems. And while everyone wants their cars to be as dynamic as their phones, nobody wants to sacrifice quality, particularly as it relates to safety.

While there's no silver bullet for maintaining quality while cutting development cycles, Qt has a powerfully expressive framework that makes it easy for developers to create, test, and debug their code in record time. There's also a large number of tools in the Qt and C++ development process that allows developers to find and fix memory problems, race conditions, and performance hogs as well as static analysis tools to pinpoint latent bugs. Combined with the ability to shortcut the design process (which we'll address in detail later), these features make Qt a natural choice to help release solid software even faster.

COTS and open source software

For the same reason that build-to-spec designs are being replaced with standardized platforms, it rarely makes sense for developers to reinvent the wheel when they need a new capability. This significantly adds to the development cost of a new application, and introduces bugs and inefficiencies that others have already fixed. However, the blend of software provenance has historically been a challenge for an automotive industry that wants to clearly assign liability and responsibility to third parties – causing much software to be rewritten needlessly.

Today, nearly every automotive project needs to deal with the fact that their final software images will contain a broad mix of third party libraries, components, and applications that are either commercial off-the-shelf (COTS) or open source.

The Qt Company provides flexible licensing so OEMs can be comfortable selecting a Qt-based project. Additionally, the Qt framework is pre-integrated with many common license compliancy tools so automakers can more easily ensure compliance.

Bill of materials (BOM) cost

No matter how specialized the discipline, cost concerns will never truly go away. However, global markets continue to shrink the planet, letting prices of products from low cost countries be seen alongside traditionally higher-margin geographies. This means continued pressure to contain and lower costs to compete in a global market.

Qt compares extremely favourably to other frameworks in terms of creating optimally performing code and minimally sized data. Meaning that selecting Qt could in fact allow you to use a lesser powered CPU, fewer RAM chips, or a smaller flash disk than if your application was created using a more wasteful framework. And because the software is hardware independent, automakers can use the same core Qt platform regardless of whether they're building a premium or entry-level vehicle – adapting the feature-set and user interface appropriately.

Doing more with less

Automakers are always very concerned with the price of components – with an average 30,000 parts per car, an extra cent here or there quickly adds up to real money.

Because Qt (with the Qt Safe Renderer) supports technologies like mixed functional safety levels and virtualization, it's now practical to drive multiple screens and functions with a single powerful CPU. This allows for a one-box solution (which can replace several discrete components) to drive an instrument cluster, infotainment, and rear-seat display – saving cost and overhead.

Cost of variance and maintenance cost

Since the vast majority of car software is outsourced, keeping careful tabs on project overruns and delays is critical to keeping the car released on time and on budget. With so many tools and processes involved in building automotive software, it can also be difficult to efficiently outsource. And as OEMs shift to more reusable software, the ease of software maintenance is also becoming an important factor.

As a product with over two decades of maturity, Qt is a very stable and reliable product. There are many sources of Qt-specific help and support available, as well as over one million Qt-trained developers – all things that help ensure that Qt-based products will be built correctly and on-time, and easily changed later if need be.

Streamlining UX design

As we've read, there's a lot of pressure to shorten automotive product development. A significant part of this reduction can come from improving the workflow between designers and developers.

Designer concerns

An automotive UX designer must balance many factors in constructing an in-car interface:

- Functions must be readily usable in short bursts
- Fonts must be readable at a quick glance
- Color palettes must support clearly distinguishable shapes in near darkness and strong sunlight
- Standard idioms that convey meaning must be used
- Animations can't distract drivers from driving
- Content must be understood properly regardless of viewer's age, language, or culture
- Graphics must complement vehicle design
- Overall look need be attractive and aesthetically pleasing

To achieve all of this, designers follow corporate policies and human factor standards in their designs. They also often perform a great deal of usability testing with real customers to see how their designs fare in the real world. A brand-new design can require many iterations before completion.

Developer constraints

Software engineers have their own stringent demands in building an in-vehicle system:

- Requirements must be implemented as precisely as possible
- Product must work consistently and reliably under all conditions
- Software ideally has no customer-impacting bugs
- Code runs well on chosen hardware platform with fixed CPU, memory size, and flash size
- Animations and transitions must be quick and smooth
- Project work can be distributed among several global teams
- Development time must be as short as possible
- Source code must be easy to understand and maintain

When software engineering teams and UX designers work together, they often find their respective requirements are at odds. Worse yet, they both depend on each other to complete their work but unrealized portions of the other team's design can introduce significant impacts. Hence the traditional interaction has been anything but straightforward.

The designer/developer workflow

In the bad old days of in-vehicle displays (early 2000s), automotive UX designers would perform usability testing with mocked-up Adobe Photoshop screens running on a laptop awkwardly balanced on a passenger seat, or perhaps on a desktop that was even more remote to the driving experience under test. They would then hand over a stack of printouts to the software team, asking them to re-implement the UX designs in C code with primitive line and box APIs.

While things aren't quite that primitive today, designers and developers have always preferred specialized – and nearly always incompatible – tools. Even with conversion tools that convert Adobe Photoshop or Illustrator files into source code, designers and developers don't speak the same language, and the resulting painful round-trips between the two domains add strain and delay to an already time-consuming process.

Reimagining the process

The design/developer problem has been one that has been largely solved by Qt 3D Studio, which consists of designer friendly tooling with a Qt integration and runtime component. This powerful tool – originally contributed to Qt by graphics experts Nvidia – brings state-of-the-art 3D tooling to the Qt ecosystem. It can easily create user interfaces in 3D and 2D (or a combination) using concepts, methods, and workflows with which designers are familiar.

Because Qt 3D Studio generates interface code that works seamlessly within Qt, developers don't need to do any work to turn a designer's UI into production code. As soon as the UX folks are done with their design, it's executable. All that remains is for the engineering team to add the functionality that hooks it into the rest of the software stack.

Enabling distributed development

The engineering workforce needed to build automotive infotainment systems is large, and it is usually broken into teams distributed across the globe. That makes the integration and coordination effort difficult. With third parties and out-sourced engineering contributing to mainline code and automotive apps becoming commonplace, the coordination difficulties increase.

These challenges were anticipated in creating the Qt Automotive Suite. It allows for one-button packaging of custom SDKs so that third parties and off-site teams can receive cleanly contained and installable snapshots in order to start contributing to the development effort immediately.

Developer ease

Developers are being asked to deliver more than ever before, faster than before, and of course with fewer bugs. Efficiency is at a premium.

Qt code doesn't write itself. But because Qt is expressively powerful, it allows developers to program with concise statements, saving time and brainpower.

Developer scarcity?
If you're worried about finding developers familiar with Qt, don't be: More than a million developers in more than 70 industries currently use Qt.

The Qt Company