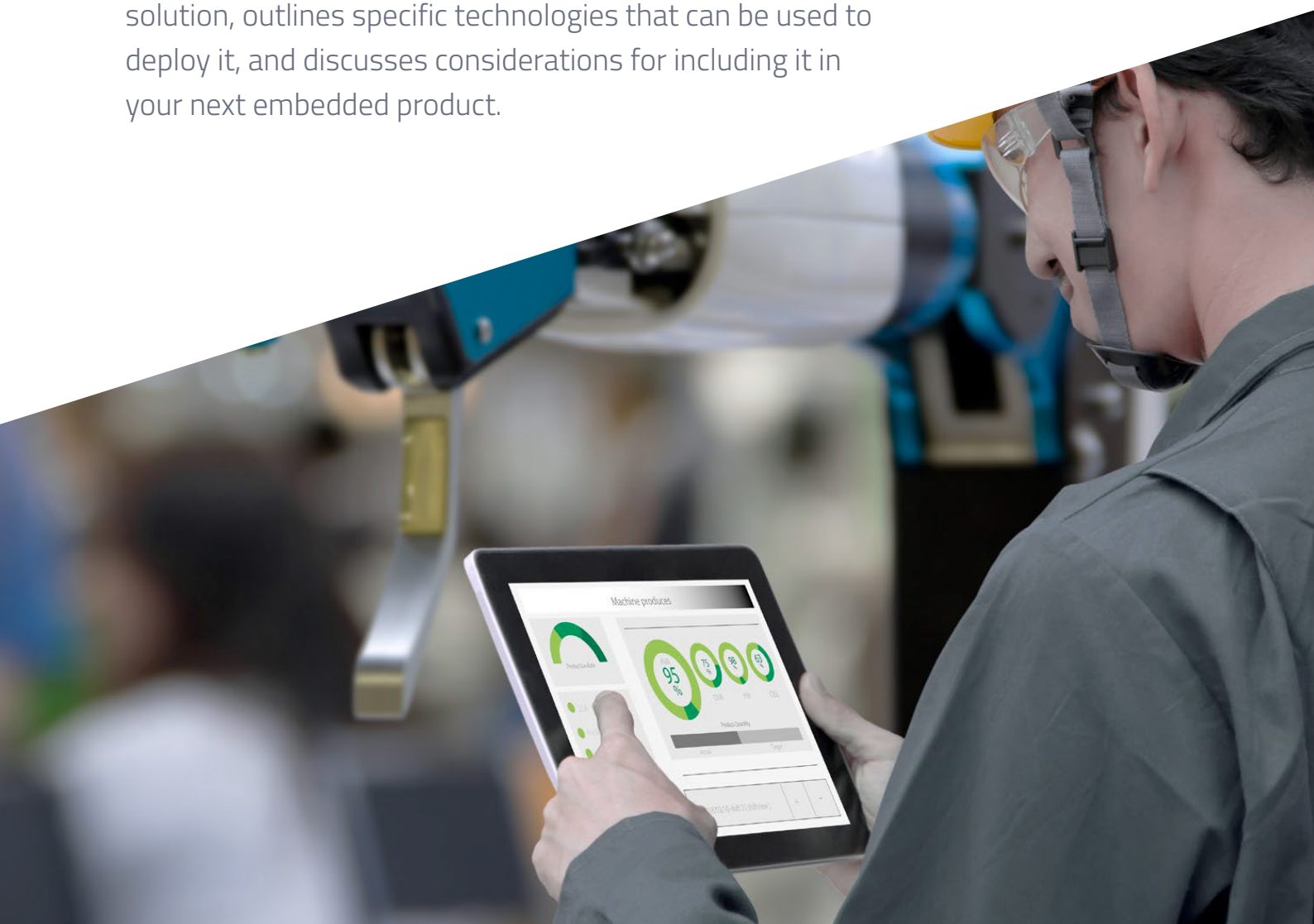
The Qt Company

WHITEPAPER

The Embedded Developer's Guide to Zero Installation

This whitepaper explores how zero installation has evolved to be what it is today, describes the components in a typical solution, outlines specific technologies that can be used to deploy it, and discusses considerations for including it in your next embedded product.



Contents

Introduction and Terminology

Zero installation allows the user of an embedded device to run applications without having to download or install them because the applications are browser-based. By eliminating the typical installation process, the user benefits from a streamlined out-of-the-box experience. And since browser technology is fundamentally standardized, a single application can run on any client device – Windows, Mac, or Linux desktops, Android or Apple phones, or any number of tablets.

Zero installation allows the user of an embedded device to control the device through a smartphone, tablet, or desktop using a web page rather than having to install an app.

Zero installation is not a specific software-packaging and distribution tool such as 0Install – despite the similarity in name – but rather a generic term for browser-based applications. While there are several definitions in use, for the purpose of this paper we consider zero installation to be a method of controlling an embedded device through a browser-based companion program hosted by the device. (Later in the paper, we also discuss an alternative zero-installation configuration where a cloud-hosted application can drive an embedded device.) Regardless of definition, there are a number of components to any zero installation solution:

Embedded device – There is always a piece of hardware in a zero installation solution, whether it is a smart home control center, an industrial robot, or an X-ray machine. The embedded device must communicate to the user about how to control the device and allow the user to view its status.

Connection – The embedded device has to provide some form of TCP/IP connectivity through WiFi, Ethernet, or cellular data. For IoT devices this may be through a full Internet connection although that's not the only method. A dedicated or direct network connection that doesn't pass through the Internet is another alternative, which may be more desirable in high security or functional safety applications.

User device (or client) – A client device is required in order to view the remote user interface such as a desktop computer, laptop, smartphone, or tablet. This client needs to be able to run a modern browser with enough free memory to load the remote application.

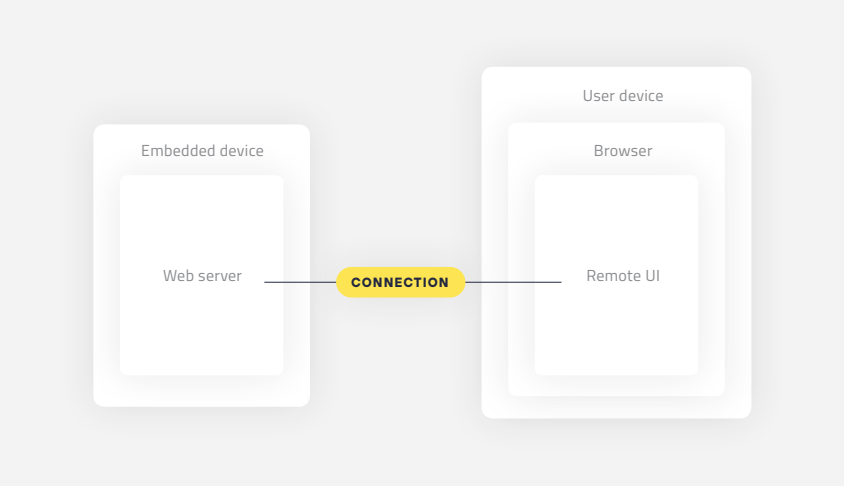
Remote UI (or remote application, or companion app) – A piece of software must run within the user's browser to control the embedded device. This is often the device's main user interface, although it may possess a physical display and buttons as well.

Browser – There needs to be a web browser running on the client device to provide the virtual environment in which the application runs.

Web server – Software is needed on the embedded device to deliver the application to the browser.

Zero Installation in the Real World

Zero installation technology provides many advantages for the user as well as the manufacturer. For this reason, it has been successfully applied in a large number of diverse areas:



Zero installation core components.

Industrial automation – Several companies are using zero installation solutions in their process control or automation control software. Bosch is a good example as they use it in their assembly line calibration equipment.

Automotive – Zero installation capabilities within vehicle modules allow dealers and mechanics to easily diagnose problems.

Healthcare – Many leading companies use zero installation solutions for medical image viewing, patient record management, and diagnostic sharing.

Smart Home – Smart home hubs allow technicians to connect and configure a home system with a more advanced UI than what is available through the user app.

Nearly any product that has an advanced embedded system with the need for users (or operators) and technicians to monitor, control, or diagnose it is a candidate for zero installation. This includes advanced manufacturing, aerospace, agriculture, construction, consumer electronics, medical, resource extraction, robotics, transportation – the list is nearly endless.

Why Modern Zero Installation Is Different

You may be old enough to remember when a new printer included an installation disk with configuration software or drivers. Although the extra installation step was always an inconvenience, it was especially annoying in an IT setting where one piece of hardware was shared among many. To help alleviate the major pain of maintaining software for every employee in an organization's network, the earliest zero installation solutions involved adding web servers and web applications to shared devices such as printers and network storage devices. Due to the cost of extra storage, web page configuration features were often limited to more expensive devices. Today's zero installation solutions are different for a number of reasons.

Users – It's not just IT specialists who need to talk to their devices; depending on the device, it may be diagnostic technicians, specially trained operators, or even consumers.

Products – While zero installation was once limited to the high-end of a product catalog, connectivity has become pervasive for nearly every embedded device. Now, IoT devices with Internet connectivity and the horsepower to run a web server are equally good candidates.

Look and feel – Previous generations of embedded devices used primitive looking and clunky UIs. Web technology standards now provide a number of options with far superior performance and attractiveness.

End result – Rather than focusing on simple configuration or monitoring tasks, today's zero installation solutions are providing fully functional browser-driven apps that compete with desktop applications in sophistication and capability.

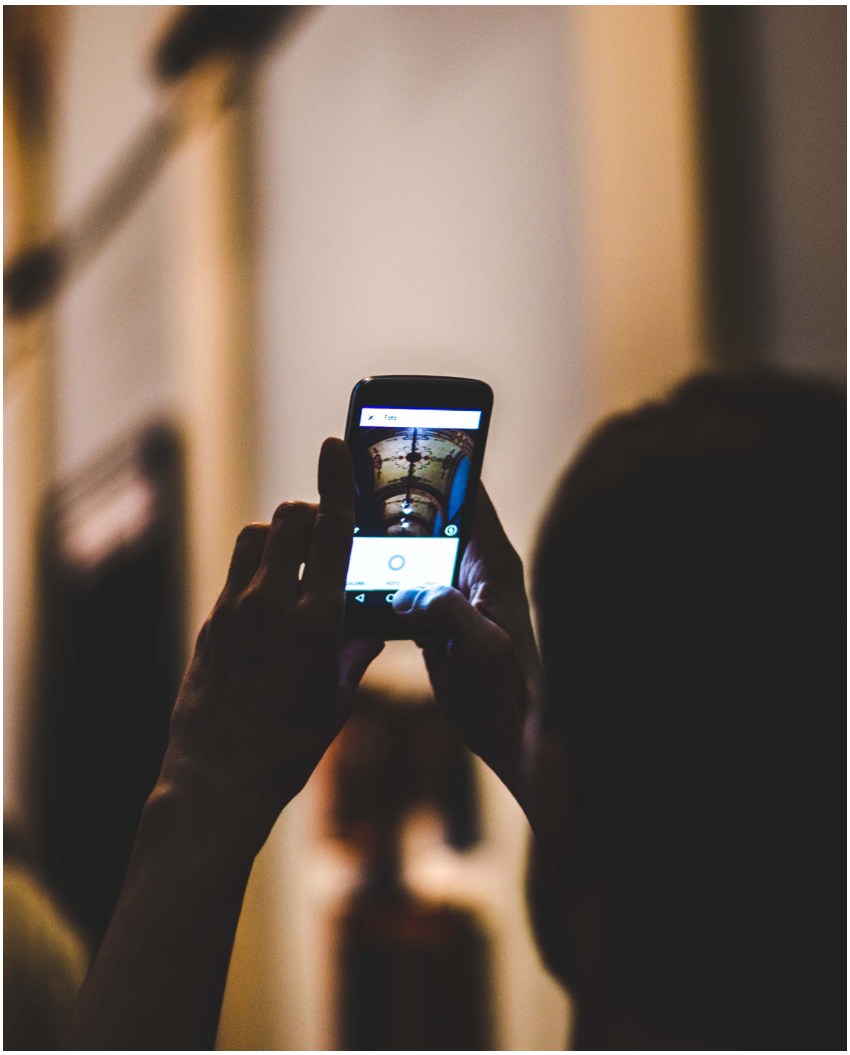
User Benefits

Today's zero installation solutions are great for the consumer as they provide a way to have "instant" access to the features of a product. The friction traditionally encountered by a user in the first few moments of using a new product is not a trivial concern; it can lead to a poor user experience, minimize the usefulness of a purchase, and even reflect poorly on a brand. The expectation for things to "just work" is part of human nature – a positive UX is a must have regardless of whether the product is built for consumers or technicians.

Previous zero installation solutions were valued primarily because they provided a way to monitor devices from a remote desktop. However, people now want to control their devices at home or on the road as well as from their desks. By allowing consumers to use their own mobile devices to control a product, zero installation gives them the freedom to roam as well as potential accessibility from just about anywhere – be it a dedicated factory computer or an Internet café.

Allowing customers to use their own devices for the UI also negates the need for additional equipment. Rather than requiring specialized tools to diagnose,

Zero installation gives user the freedom to roam as well as potential accessibility from just about anywhere.



configure, or repair a product, a zero installation solution helps users gain access to product features without new single-purpose tools.

Manufacturer Benefits

Device manufacturers have a host of reasons to build zero installation into their products. Zero installation apps are inherently cross-platform, meaning a single solution can be developed to support all of a product's potential users – instead of creating separate Windows, Mac, iOS, Android, and Linux builds. By skipping time-consuming installations while supporting the customer's platform of choice, product manufacturers are able to deliver a better, more streamlined experience that helps them differentiate their product. And there are no installation "gotchas" like shared library incompatibilities or inadequate user permissions.

When to Avoid Zero Installation

If zero installation reduces UX friction and makes more capable products easier to build – why wouldn't you build it into your next system? There are a few factors to consider.

Zero installation reduces UX friction and makes more capable products easier to build.

Non-native UI
Because a single application is served up to all clients, clearly the user interface will also be identical across all devices. While that consistency may be a benefit in many cases, it does mean that the UI will not have a native look and feel. Perhaps more importantly, the UI will also not be able to integrate any native device technologies outside the browser environment. If platform-conforming UIs or platform-specific sensors are essential to your product's definition, a zero-installation solution may not work for you; you may need to develop several platform-independent versions of the companion app.

No user interaction
Some classes of devices have no need to talk to a user directly, such as sensors or edge devices. These smaller devices are often part of a larger installation and are connected to a gateway device or a cloud application. Notably, they don't need to be individually accessible and hence don't need a UI at all. This is also true when these devices rely on standard protocols like MQTT or SNMP. The user will expect these devices to connect to existing dashboard applications that speak those protocols, so there is no additional benefit in hosting a separate zero-installation app.

No TCP/IP
A big reason to avoid zero installation is if your embedded device doesn't support WiFi, Ethernet, or cellular data. This can apply to low-power devices that use BTLE as well as devices that communicate over Zigbee or Z-Wave mesh networks.

No storage
Smaller devices or low cost devices that rely on on-board microcontroller flash memory may not have enough storage to host a web server or a web application. (Although the device itself may not have enough space, hosting the app on a cloud server may be an alternative should you really want to go the zero installation route.)

Poor connectivity
Some environments either do not have reliable connectivity or have connectivity restrictions. Whether this is due to a lack of infrastructure, security concerns, or inadequate range, you'll need to understand your device and the networking environment it will eventually live in to understand whether this may preclude a zero installation product.

Restricted device access
Depending on your deployment architecture, a zero installation UI may not have direct access to your embedded system's hardware. That means abstraction layers are needed to connect the UI and the hardware, adding development time, complexity, and latency to the solution. (Note that the Qt WebGL implementation doesn't have this limitation, which we discuss later.)

Today's Zero Installation Technology Portfolio

The often cryptic and clumsy interfaces served up by previous generations of embedded devices were limited to earlier versions of HTTP and HTML. The basic UIs of yesteryear are often no longer acceptable, especially as pervasive, beautiful, and straightforward user interfaces have driven up user expectations. Thankfully, a host of newer technology options can be used to implement a zero installation application now.

HTML5
The HTML5 markup language, especially when combined with its partners JavaScript and CSS3, has advanced significantly in capability since the early days of the web. HTML5 now is capable of fully interactive apps like Google Docs, Airtable, or Pixlr.

WebGL
For inherently graphical or 3D displays, there is WebGL, a browser-based 2D/3D graphics API based on OpenGL ES that allows browsers to display graphical content. Examples of Web GL applications are Google Maps, the Thingiverse 3D printing platform, and the BioDigital Human Platform.

WebAssembly
Web Assembly enables developers to cross-compile languages that cannot normally run within a browser (like C or C++) into a form that's compatible with a WebAssembly-capable browser. Currently, every major browser supports it: Google Chrome, Mozilla Firefox, Opera, Microsoft Edge, and Apple Safari. Although WebAssembly operates in a virtualized (and protected) environment, it aims for near native execution speed. We've got a number of examples you can look at.

The WebSocket API (WebSockets)
Many browser applications communicate with their embedded device through HTTP, although this is done by getting HTTP and an embedded application server to perform tricks they weren't really designed to do. WebSockets is a newer technology (although it's been around since 2012), which allows client-side browser apps to directly communicate with embedded server applications for better performance, lower-latency, higher security, and cleaner implementations.

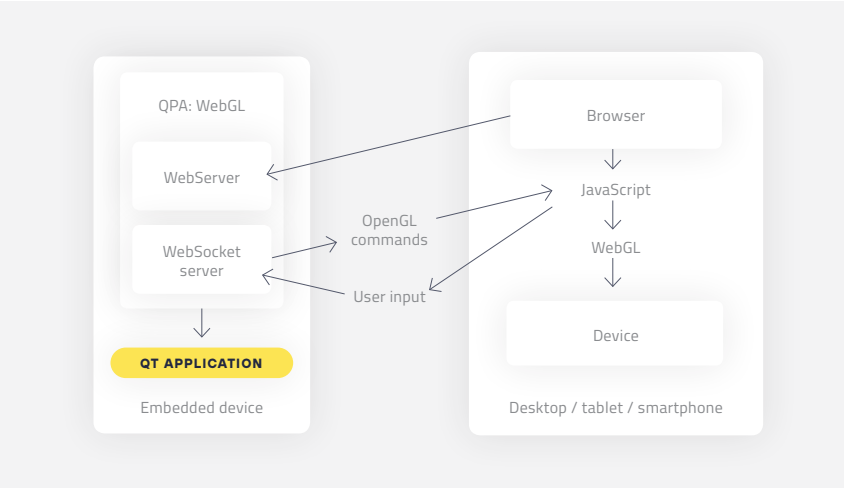
Zero Installation Solutions Using Qt

Readers of this paper no doubt already know some of the advantages of the Qt framework: it has a huge ecosystem of developers, libraries, and tools, it excels in building cross-platform applications, and it delivers the native performance of C++ to name just a few. Qt may be a very capable C++ framework, but can you also make browser-based zero installation applications with it?

Yes, indeed. There are two primary ways to convert your C++ Qt application into a browser-based version: WebGL and WebAssembly. Let's dig a little further into these two technologies to see what's required.

WebGL

The Qt WebGL solution translates Qt rendering commands into a WebGL stream that is sent to a remote browser to construct a display on an embedded device. Implementing this from a developer's perspective is trivial – in fact, in most cases, the only thing needed to make it work is adding another launch argument (-platform webgl).



Qt WebGL block diagram

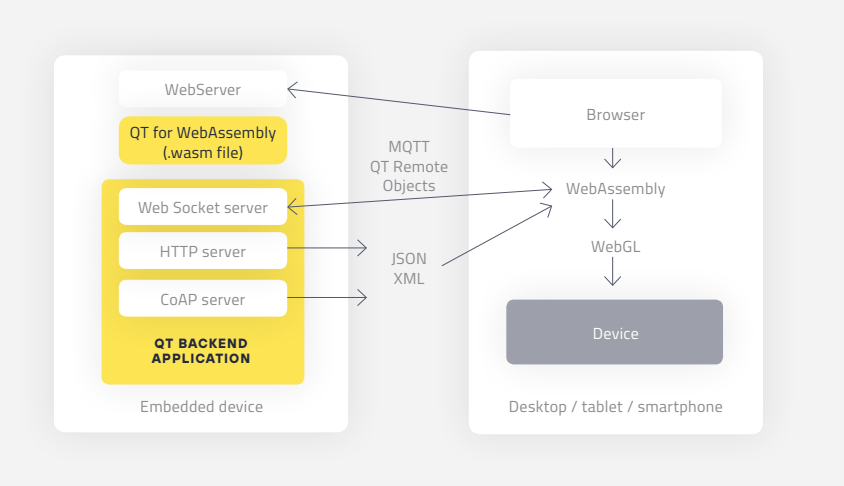
There's quite a bit going on under the covers to support this simple command as you can see in the diagram. To start, the embedded device fires up a lightweight web server for the zero installation application, as well as a WebSocket server to send and receive commands. When a browser connects to the web server, the server sends over a small JavaScript front-end application that creates WebSocket connections back to the embedded device's WebSocket server to send and receive commands.

Meanwhile, the embedded device launches the Qt application but with a QPA plug-in to intercept any OpenGL ES draw commands produced by the UI. Any draw commands are converted into a serialized WebGL stream and sent off to the JavaScript app via the WebSocket connection. The JavaScript app grabs the draw commands and uses WebGL APIs to draw the embedded device's UI within the browser. This same WebSocket channel is also used by the JavaScript app to capture the user's mouse and keyboard commands, which are sent back to the server to complete the user interface loop.

In this solution, the server side does the bulk of the work, which is different from a typical WebGL application that draws the display through client-side JavaScript programming. This makes some aspects simpler but introduces a few limitations – as detailed in our comparison on the following pages.

WebAssembly

If you choose to go the Qt WebAssembly route, a standard Qt application is recompiled with the Emscripten compiler to create a WebAssembly binary. This is delivered to the client and run in a container within the browser. Since the client's browser provides the horsepower to run the app, it's much easier to support many simultaneous clients with this approach.



Qt WebAssembly block diagram

The block diagram of the WebAssembly solution is structurally similar to that of WebGL. WebAssembly also requires a lightweight web server to serve up the application binary as well as a server to handle connections to the client.

Unlike the Qt WebGL solution, the programmer must specifically manage interaction between the application and the embedded device. One way to solve this is to use a micro-service architecture that provides fine-grained modular services that handle each specific task that the client app needs to execute. The client's browser application packages up any requests and sends them to the embedded device. The device's server receives the request packets, unwraps them, and executes them. (MQTT or JSON are two lightweight formats with corresponding Qt classes that make them both easy to use.) Another alternative is QtRemoteObjects, which allows the Qt slot/signal mechanism to operate between machines.

Comparison of WebGL and WebAssembly

	WebGL	WebAssembly
How it works	Device sends serialized WebGL data stream to client browser to draw remote UI	Device delivers WebAssembly binary to client browser; app in turn runs within browser
Converting existing Qt apps	Trivial. Client-side application graphics are handled automatically and direct device control is unaffected	Simple. Conversion is mostly handled with a recompile using the WebAssembly toolchain. Components that directly access hardware need to be converted to use microservice architecture (or equivalent)
Application speed	Dictated by embedded hardware	Dictated by client device hardware
UI responsiveness	UI runs on embedded hardware and is transferred to client device so responsiveness is dependent on sufficient network bandwidth and low packet latency	UI runs directly on client device so responsiveness is dependent on client hardware specifications and other client applications running in background
Initial startup time	Quick – nearly instant. Client browser only needs to receive small stub before displaying first frame	Longer – may take a few seconds. Full WebAssembly application must be transferred to client device and compiled by browser on first launch
Number simultaneous clients	One. Multi-process Qt WebGL is on the roadmap	Unlimited
Difficulty of reverse engineering	Client-side reverse engineering is impossible as client device only receives WebGL data stream	WebAssembly code that is transferred to client device can be reverse engineered, although it's much harder than with the HTML5/ JavaScript equivalent
Limitations	Not applicable if app uses non-OpenGL drawing (e.g. Qt Widgets)	
	Not applicable if app includes third party modules without source, since all code must be re-compiled into WebAssembly	

Qt Zero Installation Technology Stack

So what do you need to build a zero installation solution with Qt?

First, you'll need Adobe Photoshop or Adobe Sketch for your designers to create graphical assets. You'll also need Qt Design Studio, which is used to create declarative QML from the imported graphical assets for your application's various screens.

Then, you'll need Qt Creator to provide the necessary compiler, tools, and libraries to build Qt applications. This integrated development environment (IDE) can be used to build Qt applications from your designer's graphical assets using either QML or C++. (Note: Qt for WebAssembly requires that you select the option to download Qt sources so they can be recompiled for WebAssembly.)

You'll also need Qt for Application Development, which provides a rich set of components and tools for building UIs (QML, Widgets, OpenGL, SVG, Qt 3D), managing hardware communication (networking, WebSockets, Bluetooth, serial ports, CAN, Modbus, sensors), and taking care of data management (SQL, XML, image formats).

Next up is Qt for Device Creation. This package supplies the necessary components for building, debugging, and deploying on embedded devices. It includes cross-compilation toolchains, software for debugging over USB, the Boot to Qt software stack, Qt for RTOS, and a number of reference target platforms. It also includes both the Qt WebGL and Qt for Web Assembly components, enabling developers to create either type of solution.

Finally, one of the core parts for any Qt zero installation solution is Qt for Automation. That's because Qt for Automation includes a number of key pieces to communicate between embedded devices and client apps such as Qt MQTT, Qt OPC UA, Qt KNX, Qt CoAP, and Qt Remote Objects.

Zero Installation and the Cloud

Another option to hosting a zero installation application on an embedded device is to have it hosted in the cloud. Qt gives you the flexibility to include this approach in your strategy, as all of the tools remain the same. A cloud-based app is a good approach if your embedded hardware doesn't have the horsepower to host or run complex, data-hungry apps or if the app needs access to data resources maintained in the cloud.

The downside to a cloud-hosted option is the connection. When the embedded device hosts the client app, a connection between the client browser and the device is guaranteed. A cloud-based architecture necessitates that the client machine is connected to the Internet with no interference from proxies, VPNs, or firewalls, and requires a separate connection from the client browser to the embedded device. These requirements aren't unsolvable but can slightly complicate the "no friction" promise.

One big upside to zero installation in the cloud is that you can merge input from many devices across a network into a single model, critical in the development of digital twins. A digital twin takes sensor inputs from a system or object that exists in the real world to create a real-time virtual replica of it. This allows the object to be remotely examined for proactive maintenance, operational efficiency, and educational information. Digital twin technology is starting to see traction in manufacturing, healthcare, automotive, building automation, space, and defense.

Functional Safety and Cybersecurity

How does zero installation fare when there are cybersecurity concerns? Many IT organizations use company firewalls and user permissions that would need to be loosened to install application software. Since zero installation apps run in a browser, they allow users to access the product's companion apps while maintaining strict IT controls. The browser provides a virtualized, contained environment, keeping the client computer isolated from attempts to hack in at the client end. The connection between the hardware and client can include secure sockets for encryption and authentication of the data passed between the two machines.

Zero installation gives users many advantages with a no hassle start, lower maintenance, and cross-platform and mobility capabilities.

Zero installation also helps prevent reverse engineering. In a WebGL implementation, reverse engineering from the client-side is impossible, since no application data is sent – only WebGL rendering commands. A WebAssembly application does send an executable to the client. However, the resulting binary code is far less comprehensible than the equivalent human-readable HTML5/JavaScript app would be.

Qt can be used to create industrial (IEC 61508), automotive (ISO 26262), and medical (IEC 62304) certified systems using Qt Safe Renderer. However Qt zero installation using WebAssembly doesn't need any type of rendering on the embedded system at all. It's a headless solution, making it more straightforward to certify. (Unfortunately, the same simplicity cannot be said for the WebGL variant, which would not be easy to certify.)

Conclusion

Zero installation gives users many advantages with a no hassle start, lower maintenance, and cross-platform and mobility capabilities. Product developers often choose Qt to create these solutions as Qt gives them the ability to create a superior user experience while shortening and simplifying the development process with cross-platform flexibility, reliable and performant tools, and a broadly supported developer ecosystem.

Another piece of good news is you can reuse your existing native Qt application in a browser-based zero installation app with either Qt for WebGL or Qt for WebAssembly. Besides ease of implementation and a single code base, a Qt-based zero installation application may have additional advantages, such as improved security and performance.

If you think zero installation might be the right solution for you, contact us for help understanding how to implement it into your future products.