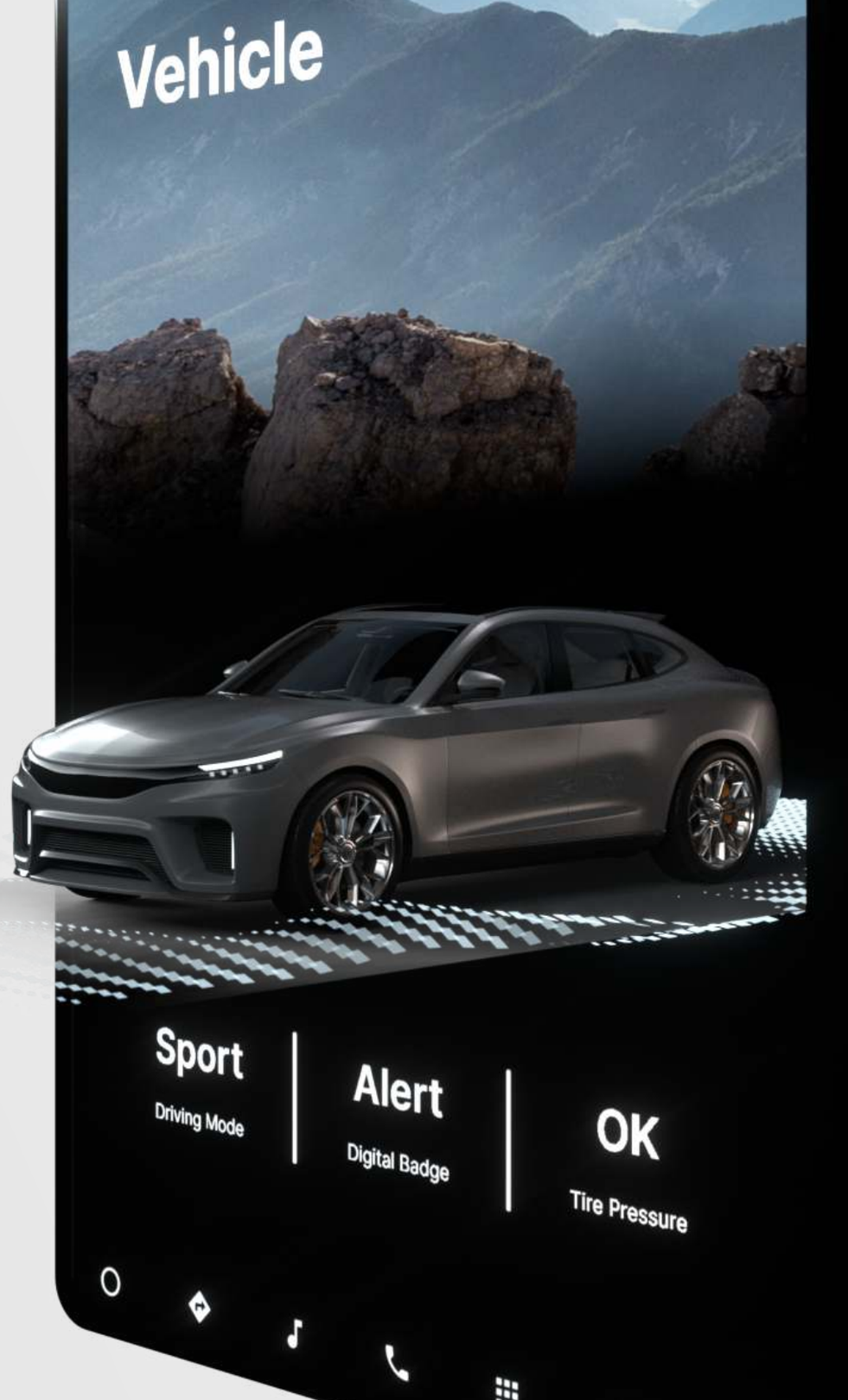
RIGHTWARE

A New Standard for Android Automotive HMI Creation



KANZI® ONE

Table of Contents

The Android Automotive foundation

Android™ is an established, robust, and cost-efficient operating system, with a large pool of users and developers, offering modern cars a broad network of services and forward-looking functionalities.

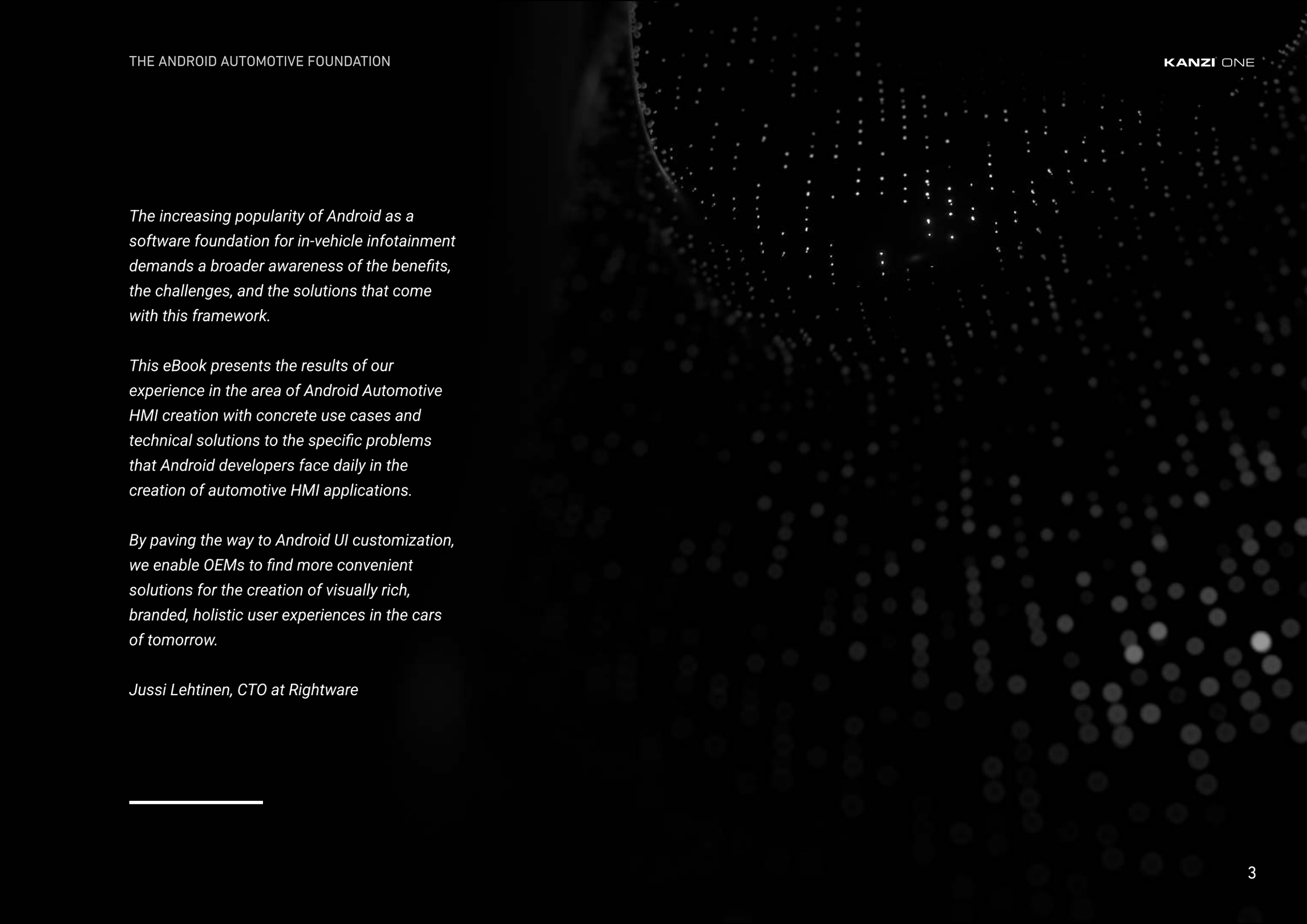
A growing number of automakers are adopting Android Automotive as the foundation for their infotainment systems, as the platform grants carmakers several out-of-the-box benefits—like connectivity services, multi-display support, maps, media players, Bluetooth, etc.—essential to the modern digital cockpit.

Despite the advantages, HMI developers know how complex it is to build an HMI that runs on Android and interacts seamlessly with this framework, no matter the HMI toolchain used.

At Rightware, we have tackled the complexity of building fully customized Android home screens and launchers for over a decade and have isolated the primary areas where traditional HMI frameworks fail to integrate with the system.

While the results of our efforts help OEMs leverage and customize the Android OS, they are also relevant to OEMs aiming to bring consistency and seamless integration across all their vehicles' screens regardless of operating system, from IVI to cluster, HUD, and rear-seat entertainment.

Carmakers will find herein the recipe for creating a fully integrated multi-screen intelligent cockpit experience for the vehicles of tomorrow, with technology available today.



The increasing popularity of Android as a software foundation for in-vehicle infotainment demands a broader awareness of the benefits, the challenges, and the solutions that come with this framework.

This eBook presents the results of our experience in the area of Android Automotive HMI creation with concrete use cases and technical solutions to the specific problems that Android developers face daily in the creation of automotive HMI applications.

By paving the way to Android UI customization, we enable OEMs to find more convenient solutions for the creation of visually rich, branded, holistic user experiences in the cars of tomorrow.

Jussi Lehtinen, CTO at Rightware

The need for a robust in-car software foundation

Software has become an integral element of the in-vehicle experience. Modern cars are increasingly equipped with powerful visual tools to support drivers on their journeys. Analog gauges and needles have been replaced by more enticing digital implementations, and the place once reserved for the radio is now occupied by rich, interactive infotainment systems. Here, high-tech devices offer touchscreen navigation, voice control, and connectivity to local and remote services, transforming modern cars into an extension of the driver's digital experience. The driver's perception is augmented by advanced driving assistance systems where bleeding-edge graphics and hyper-realistic rendering of the car in its real-time surroundings enhance situational awareness, provide alerts about points of interest, or warn of potential threats.

The carmaker's needs

Reliable operating system

Rich functionality

Brand differentiation

Simple workflow

Rapid time-to-market

Contained costs

The implementation of such a wide range of functionalities requires automakers to choose carefully the software and hardware infrastructure powering their "intelligent cockpit."

For OEMs, brand differentiation through awe-inspiring Signature UIs and innovative app design must not compromise the development workflow, let alone increase time and cost.

The Android infrastructure

A robust, cost-efficient foundation for automotive applications

Clearly, developing a custom, proprietary operating system from the ground up is expensive and time-consuming, not to mention the effort to establish an entire ecosystem, app store, and companion apps to deliver tangible value to the end user. Such an investment needs to be considered carefully, especially when robust, cost-efficient software solutions are ready to be adopted and incorporated into cars.

Android is on the rise again.

Not only has Android grown into a mature OS supported by most hardware vendors, it has also evolved to include a dedicated extension for the car industry—Android Automotive—optimized to work as a built-in platform for automotive infotainment systems.

Multi-display support

UI framework

Bluetooth stack

Media player

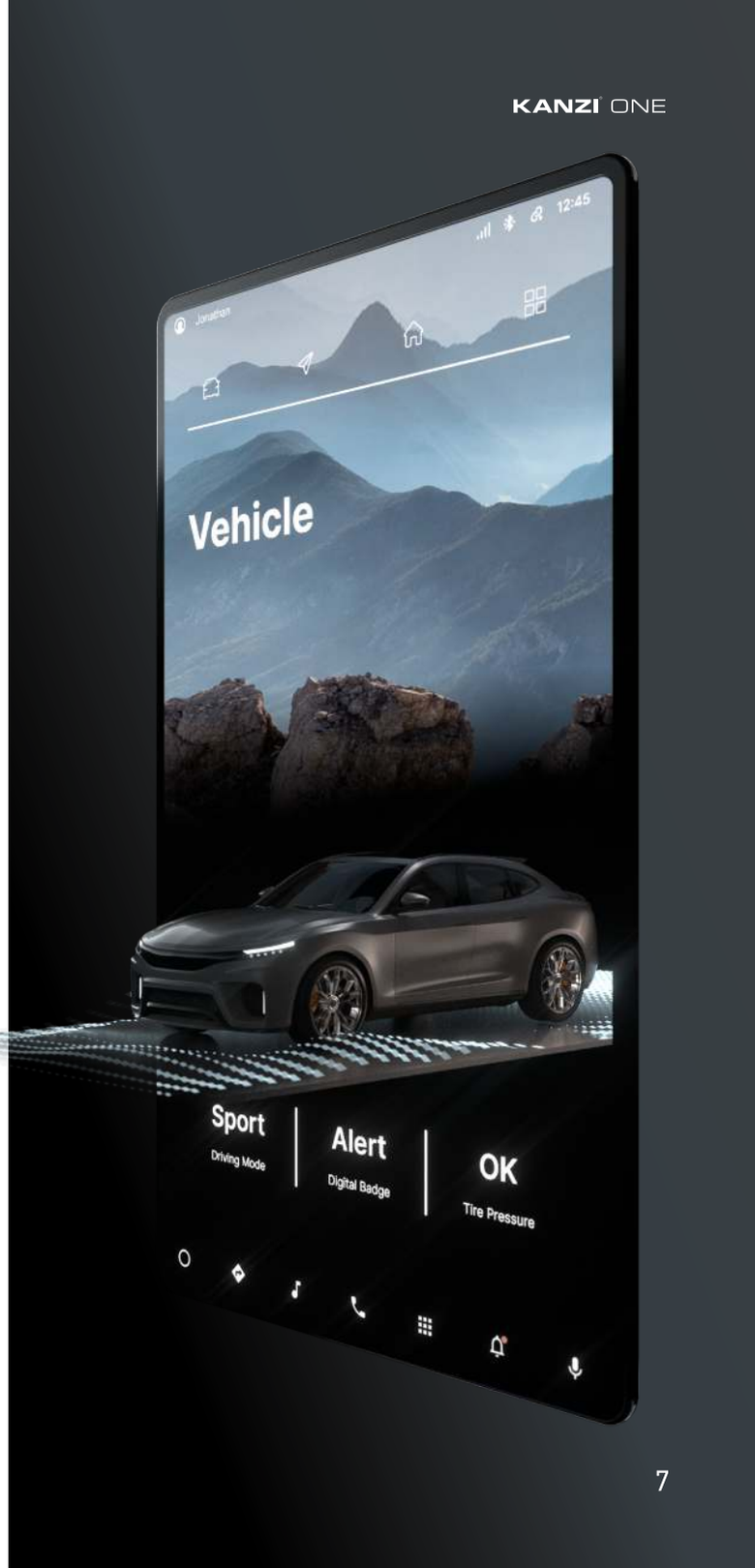
Google Automotive Services, including:

- Google Maps
- Play Store
- Google Assistant

Android Automotive comes with a host of inherent benefits for setting up the in-vehicle infotainment system.

The up-front investment when using the Android codebase is contained, and its use does not require a commercial arrangement with Google. However, Google does provide Google Automotive Services (GAS), precompiled and licensed directly to OEMs. Recognizable examples are Google Maps, Play Store, and Google Assistant.

Several car models already use Android as their in-car OS, and we will continue to see more deployed Android versions on the road in the coming years.



A brand's own Signature UI on Android Automotive

It is hard not to see the convenience of adopting this powerful infrastructure offering such a variety of ready-made services. However, just as smartphone producers invest in adapting the system to their brand and customer base, carmakers may not be satisfied with the default Android offering.

Quality of graphics

Utility of applications

Reliability of services

Holistic look-and-feel

In modern cars, the digital component is so important that the utility of applications and the reliability of services define the functionality of a car. At the same time, the prestige of a brand and the satisfaction of its customers also draw on the quality of graphics and on the alignment with the look-and-feel of the brand itself across all screens of the vehicle.

This is what we in Rightware call a Signature UI, something far beyond the default Android offering.

This level of user experience requires a dedicated HMI framework enabling powerful graphics and automotive-grade functionality.

Challenges in Android customization

Building visually rich applications and UIs on Android, customized with the typical blend of 2D and 3D elements that are valuable for in-car infotainment and driver support, can be a daunting task.

All graphics frameworks used to create automotive HMIs face serious challenges when operating on the Android infrastructure—at least, until the release of Kanzi One.

Let us explore why this is the case and what is at stake.

01 Multiple codebases

Different languages for HMI frameworks (C++/C#) and Android (Java)

Communication based on complex JNI bridges

Difficult and error-prone code

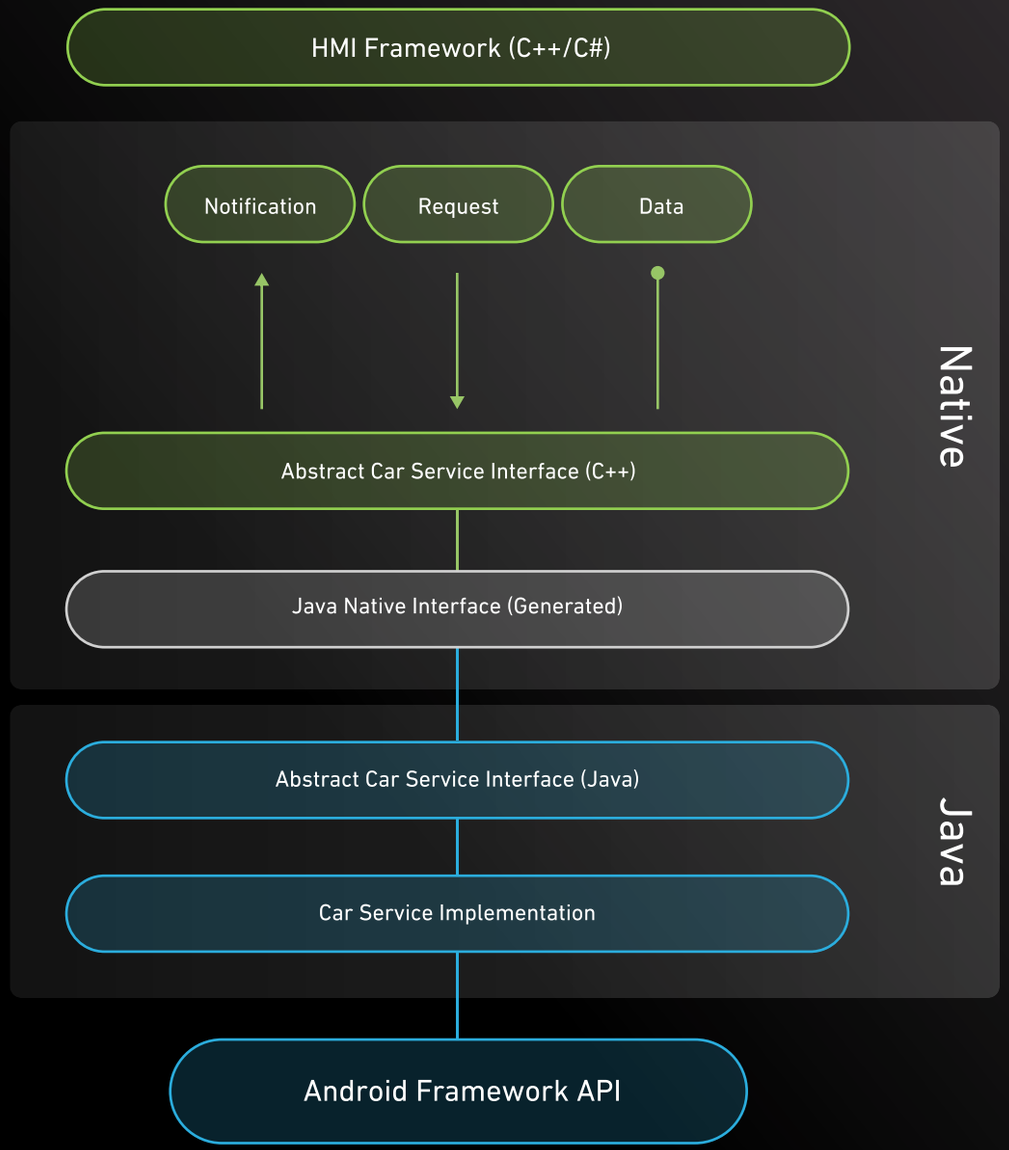
The first major challenge comes from the different codebases used in the HMI framework that powers the user interface and in the underlying Android system. The Android SDK and apps use Java almost exclusively, while HMI frameworks are typically written in variants of C (C++ or C#).

Interfacing the two codebases is a complex job. Not only must code be written in two languages, but extra code is needed to enable communication between the two frameworks.

This complexity is evident in the following diagram: several layers of code are needed to connect the Android framework API to the UI elements powered by the generic HMI framework.

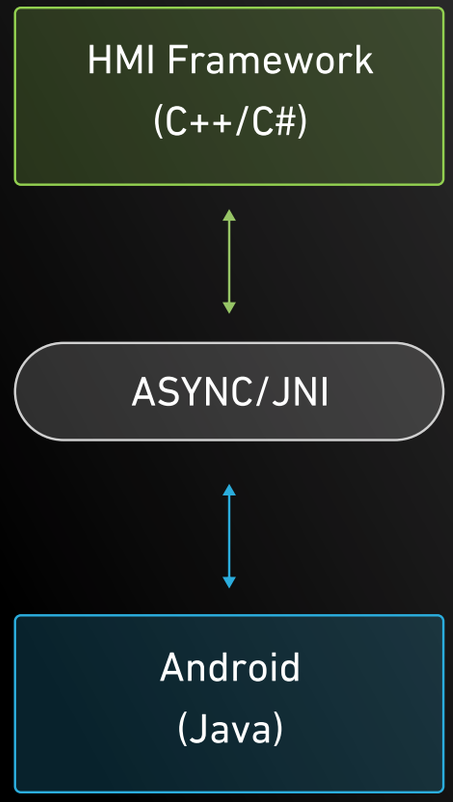
JNI bridging code is known to be difficult, error-prone, and tedious, especially when dealing with custom data structures that need manual serialization. In other words, the programmer should have strong skills, experience, and perseverance to cope with such a job.

It should also be noted that the in-between glue-code may need to be changed in line with any change to the outer layers, adding repetition and complexity to the task.



02 Separate threads

The UI and the underlying Android application run in separate threads, forcing communication between the frameworks to be asynchronous and requiring locking to ensure thread safety. This further complicates the workflow, is inefficient, and impacts the performance of the system.



03 Lack of a dedicated IDE

Debugging

Profiling

Emulation

The scenario is even more discouraging due to the lack of an integrated development environment allowing to debug Android apps consisting of multiple codebases. For this reason, the source of even trivial issues may take a long time and a great deal of effort to be detected.

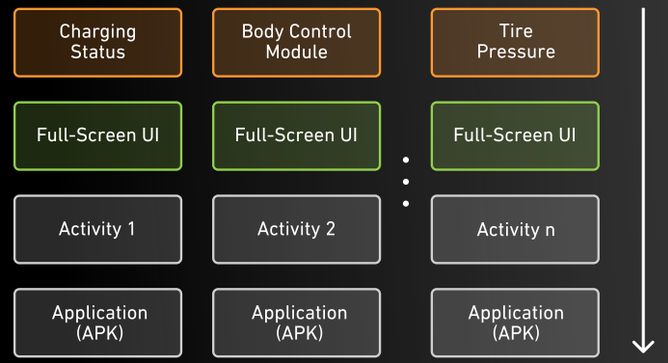
The lack of an emulator makes it exceedingly difficult to check proper app behavior on different devices, that now need to be physically available on the developer's desk.

If creating a customized UI that runs on Android is such a complex task, it's not surprising that the result of such work suffers from inherent limitations in the appearance and functionality.

04 Intent handling and resource sharing

Android applications have their own resource allocation system and a modular structure by which different activities and views within the same app share assets. This is advantageous both in terms of project structure and resource management when an application needs to handle multiple intents.

Typically, HMI frameworks running within Android cannot exploit these features. Every UI implements a full-screen activity that uses all application resources. When another activity is needed, another dedicated app must be created —no matter if these share the same resources. This leads to inefficient resource management and deviates substantially from the standard Android application structure.



05 Activity lifecycle management

Another limitation of traditional HMI frameworks is that Android activity callback methods cannot be directly exploited. For instance, backgrounding and foregrounding activities have their own lifecycles that need to be respected.

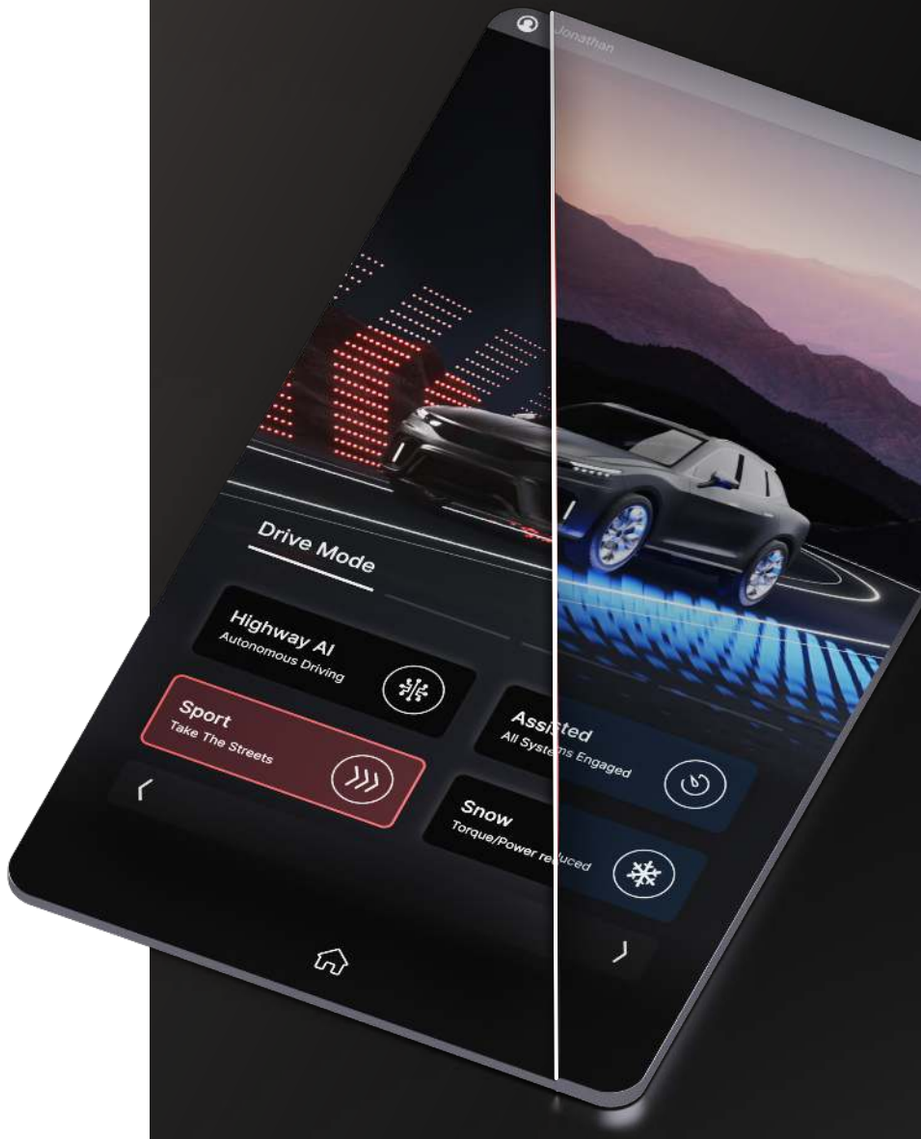
Again, custom glue-code is needed to control the activity lifecycle, and unsurprisingly, such gimmicks make it very challenging to access and control the application state.

06 UI stacking and theming

In terms of appearance, the limitations that traditional HMI frameworks face on Android are glaring. There is no control over UI element stacking, with the view frame occupying either the topmost or lowest layer of the screen.

At the same time, UI elements do not integrate with the overall Android theme and appear as separate elements on the screen.

Ultimately, UI elements created with a traditional HMI framework do not blend with those coming from Android, the ecosystem, or third parties, leaving much to be desired in the holistic experience.



07 Event propagation

An HMI layout is hierarchically structured, built from multiple UI layers occupying different areas of the screen. Again, each UI element can come either from Android or from the HMI framework.

The latter should enable propagation of touch input through the UI layers until it reaches the application it's intended for—a result not easily obtained with traditional HMI frameworks.

Better, faster, smarter with Kanzi One

At Rightware, we have tackled these complexities one by one and, with Kanzi One, we can finally state that they belong in the past.

Kanzi One opens a new horizon of creative opportunities for Android-based systems.

The new workflow tailored to the Android Automotive operating system drastically reduces the complexity of creating an advanced multi-screen UI for the modern digital cockpit.

Deep Android integration

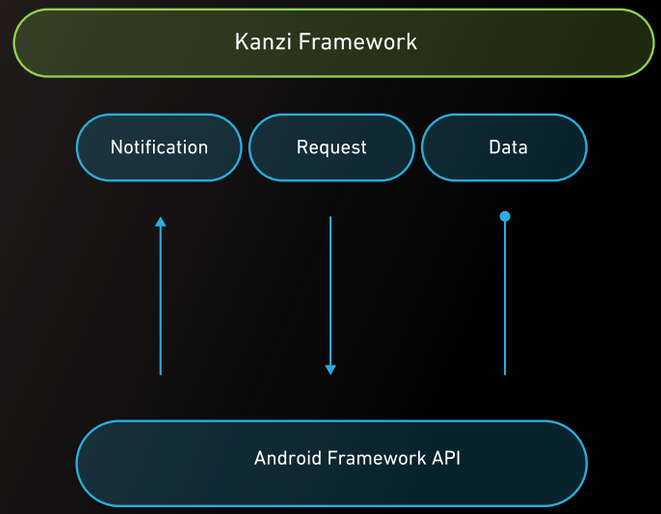
Carmakers can finally take full advantage of the Android OS and build powerful user interfaces on top of it, without limitations.

State-of-the-art graphics

Thanks to its uniquely deep integration with Android, Kanzi One sets a new standard for the creation of state-of-the-art graphics and advanced HMIs on Android.

Simplified software stack

Evident in the diagram is the vastly simplified software stack required to interface the Android system with Kanzi.



Deep Android integration

With Kanzi One, the HMI framework architecture has been renewed for full compatibility with the Android OS and ecosystem. Kanzi One guarantees powerful graphics on top of any Android functionality, without compromise.

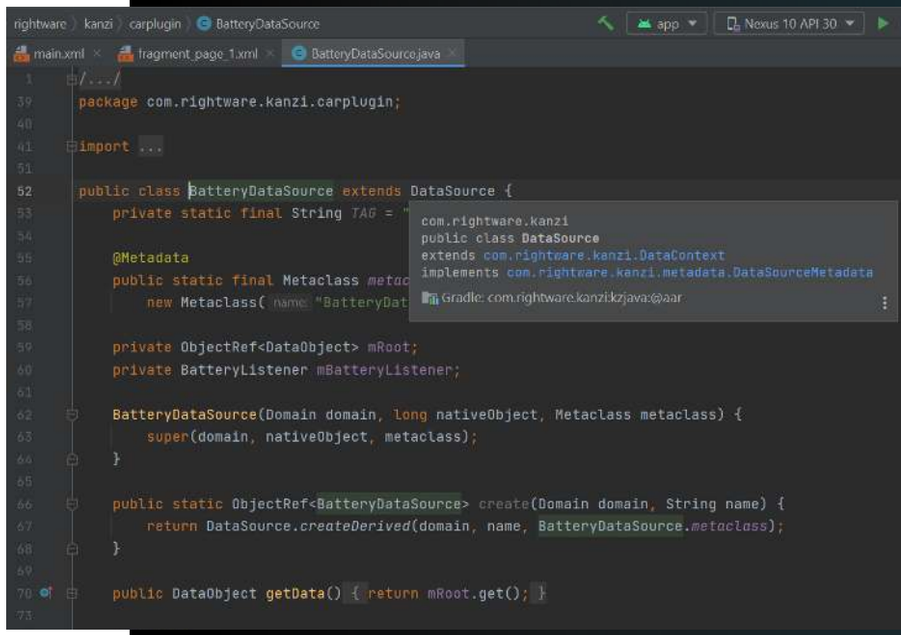
Java and Kotlin for apps and plugins

While rooted in high-performance C++, Kanzi now exposes Java and Kotlin APIs, allowing developers to create applications and plugins directly in these languages. Programmers can write code according to the standard rules, constructs, and lifecycles of regular Android apps. And they can do this using the default Android IDE for development, debugging, and profiling.

Standard Android workflow

Android IDE for development

Android developers can write Kanzi plugins in Java or Kotlin within Android Studio using the Kanzi Java and Kotlin API and export them to Kanzi Studio for data binding with the UI.



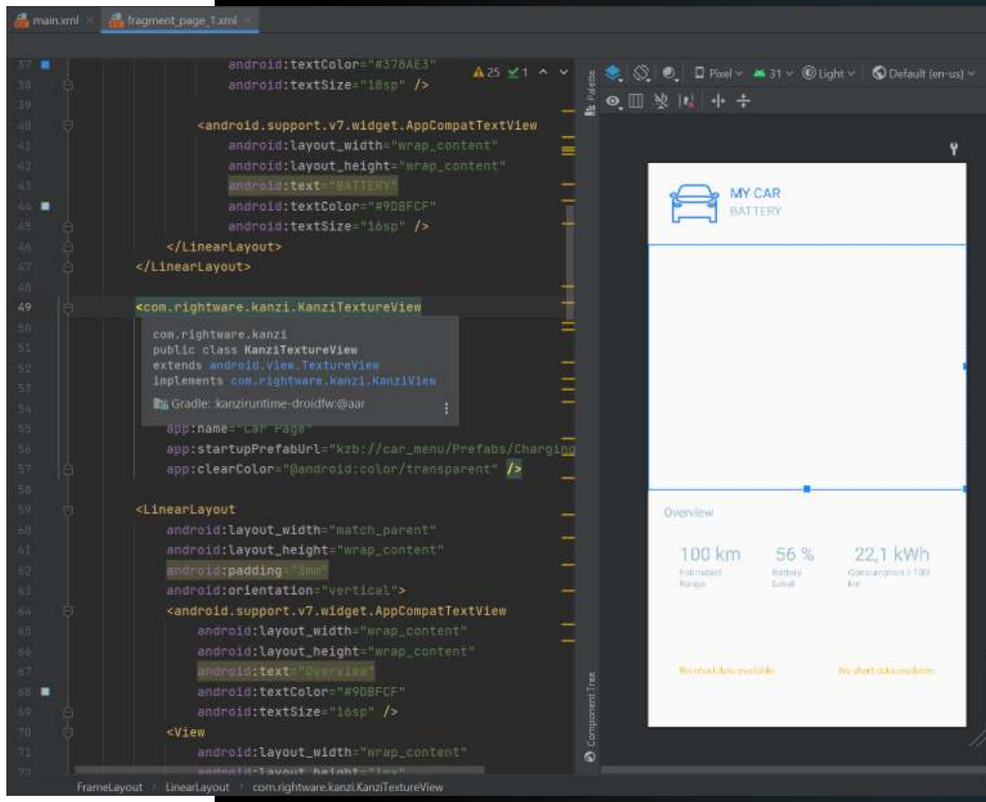
Kanzi View derived from the Android View class

- Free composition with Android views
- Multiple views per Activity
- Resource sharing across activities
- Proper touch and focus interaction
- Activity lifecycle management

Kanzi libraries and design assets can be imported into Android Studio to build new apps with the standard Android workflow and the rich graphics offered by Kanzi.

Kanzi Views are derived from the corresponding Android View class and can be used within Android apps in exactly the same way. One can compose multiple views, either Kanzi, Android, or third-party, in any order within a single Android application.

At the same time, touch and focus events interact as expected even when stacking Android and Kanzi views.

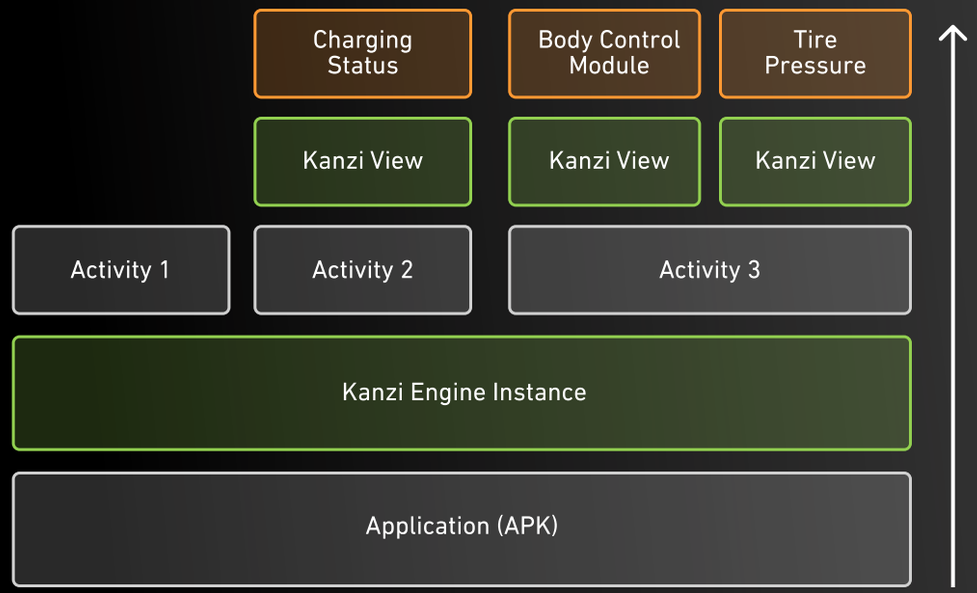


As we have seen, this kind of flexibility within the Android system is unavailable with other HMI frameworks, where typically a single view consumes the entire application's screen.

In terms of performance, all views in the same app share a single Kanzi Engine instance along with all the loaded Kanzi files and resources.

When no Kanzi View is active, Kanzi Engine can be unloaded from memory, freeing system resources.

Multiple Kanzi views in a typical Android Automotive application.



The seven key ingredients

The digital cockpit is an essential element of modern cars, not only in terms of functionality, but also as a key factor in the prestige of a brand.

A flexible, reliable, and powerful operating system offers the foundation for a sound in-car infotainment system. But to create a truly compelling user experience, carmakers need the ability to customize and add their own touch to the system, in terms of both look-and-feel and functionality.

Android Automotive, enriched with the advanced 2D and 3D graphics enabled by Kanzi One, provides the extended framework needed to create the car of tomorrow.

Kanzi One presents UI designers, engineers, and OEMs in general with seven key benefits to achieve this.

1. UI creation in the standard Android languages, so multiple codebases and glue-code are not needed
2. Ability to run Kanzi Engine in the Android UI thread, with no need for asynchronous communication and locking
3. Use of the Android IDE for development, debugging, and profiling
4. Typical Android application structure, enabling modularity and resource sharing
5. Use of standard methods for activity lifecycle and state management
6. Full control over the position and layering of the UI, and integration with the overall theme
7. Seamless touch propagation across UI layers

The multi-platform experience

We conclude this eBook with a final “bonus” benefit stemming from the overall new scenario that this Android integration opens on the cross-platform experience enabled by Kanzi.

While Kanzi’s deep integration with Android drastically reduces the complexity of creating Android-based HMI applications to an extent unparalleled by any other HMI framework, Kanzi retains its cross-platform promise.

Carmakers are finally presented the possibility to run multiscreen HMIs seamlessly on each hardware and software platform in the car—not only the Android-based ones. This improves the workflow even further, in that all graphics assets created with Kanzi can be reused, for instance, on the cluster, the HUD, and the Android IVI system, no matter the OS powering them. Thus, a consistent look-and-feel across all the car’s displays is guaranteed with minimal effort.

Kanzi multi-platform technology allows reuse of graphics assets across screens, securing:

- A holistic look-and-feel across the entire dashboard
- Faster time-to-market
- Reduced cost
- A Signature UI on every screen of the car

The Kanzi framework

In production with over 50 car brands

Automotive assets

80% UI code eliminated

Agile workflow

Fast prototyping

Easy deployment

Kanzi is a production-proven HMI framework, specifically designed for automotive use cases. It offers state-of-the-art rendering capabilities for all the car's displays with its automotive-centric graphics engine. Kanzi powers the digital cockpit experience of more than 50 popular car brands globally.

Thanks to its efficient workflow and wide array of automotive solutions, Kanzi is the natural all-in-one choice for OEMs to boost their designers' and engineers' creativity, in terms of both visuals and functionality.

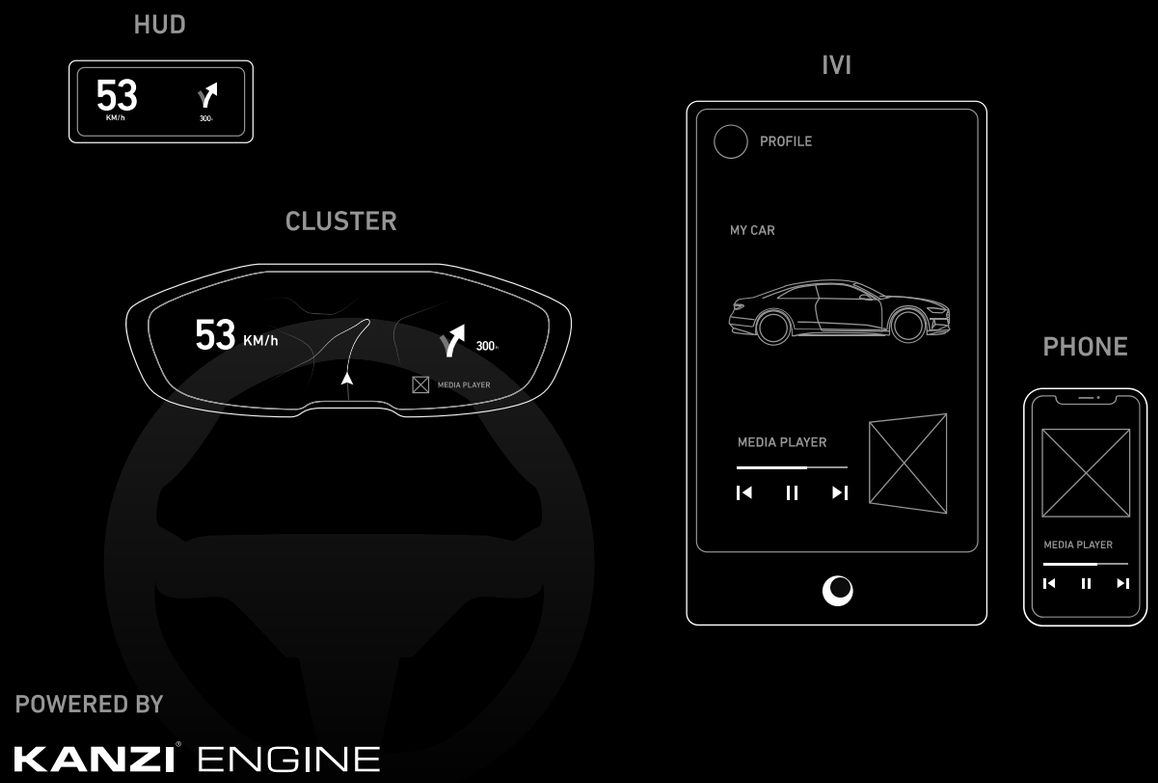
Kanzi One is a product bundle including the core Kanzi framework and a set of productivity-boosting feature packs.

Kanzi Engine

The most powerful real-time graphics engine for automotive HMIs, combining physically-based HDR rendering with a production-proven runtime designed for extreme efficiency on embedded systems.

Designed for powering automotive HMIs, Kanzi Engine offers state-of-the-art 2D and 3D graphics with fast startup time and low memory footprint.

The availability of PBR and postprocessing effects enables game-engine-like rendering capabilities for the creation of limitless compelling user experiences within cars.



Kanzi Studio

The core HMI authoring tool used by designers and developers for the creation of future-looking, multi-screen UIs features:

- Real-time 2D and 3D UI design tools
- Advanced animations and transitions
- Multi-project workflow
- Automotive asset and material library

The visual editor offers designers the possibility to create models, interactions, and animations without writing code.

Developers use the same tools to manage logic, data integration, and performance optimization in agile interaction with designers.

DESIGN WITH

KANZI® STUDIO

FEATURE PACKS

- ☑ KANZI CONNECT
- ☑ KANZI MAPS
- ☑ KANZI PARTICLES
- ☑ KANZI STEREOSCOPY
- ☑ KANZI SHAPES
- ☑ KANZI VR

Kanzi feature packs

A set of productivity-boosting add-ons to extend the Kanzi framework with the latest automotive functionality and accelerate the development process.

KANZI PARTICLES

Implement astounding effects and animations

KANZI SHAPES

Render vector shapes without resolution constraints

KANZI CONNECT

Share data, content, and services across devices

KANZI MAPS

Enhance navigation with creative visualization

KANZI STEREOSCOPY

Enable realistic depth effects

KANZI VR

New workflow accelerator shortening the design cycle and simplifying the decision-making process.

With or without the VR headset, the virtual environment enables simulation of the UI in a variety of contexts, allowing designers to rely on an immersive, real-time, interactive preview.

www.rightware.com

Rightware® is the pioneering provider of automotive graphics software tools and services.

Our mission is to help automakers become highly efficient and creative with no limits. Our vision is to transform the traditional HMI into a real Signature UI.

A ThunderSoft® company