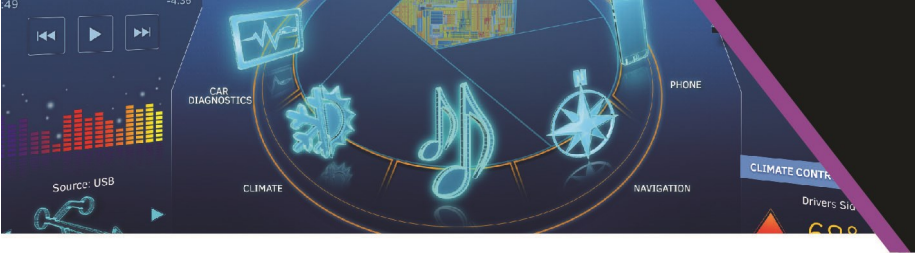


Integrated Computer Solutions

ICS

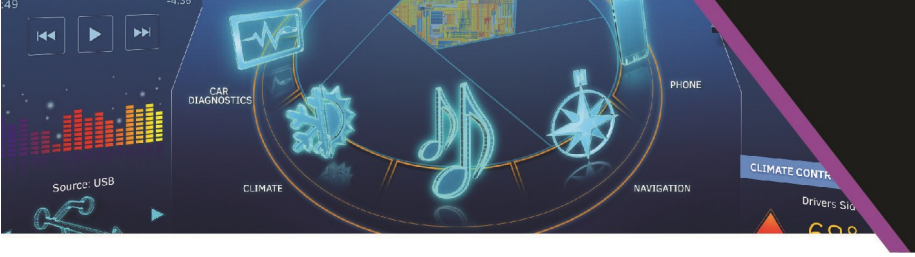
Intel In-Vehicle Solutions Proof of Concept

High Level Design Document

Integrated Computer Solutions, Inc.

54 B Middlesex Turnpike, Bedford, MA 01730

617-621-0060 www.ics.com info@ics.com



Integrated Computer Solutions

ICS

Background

Demand for connected devices in automobiles is accelerating as mobile ecosystems are maturing. The ability for automakers to differentiate in an increasingly competitive environment depends on its ability to integrate a smart in-vehicle infotainment (IVI) system strategy and deliver a fully connected lifestyle to consumers. The manufacturers who succeed at providing an open, scalable and easily configurable automotive IVI will be the future market leaders.

Consumers expect the same user experience (UX) in their automobiles as on other personal devices that blend beauty and simplicity with practical use and technology. These increased expectations for design and performance also add complexities in interoperability and safety for the automaker.

Integrated Computer Solutions (ICS) has delivered a proof-of-concept of a next generation in-vehicle infotainment system based on the Intel® In-Vehicle Solutions compute module and software foundation. ICS has a long history of advanced software development and UX design. For many years, they have been on the forefront of developing and integrating custom automotive infotainment systems that consumers have come to expect.

Two frameworks were used to deliver the ICS automotive HMI (human machine interface) solution: Intel's Automotive Services API and Qt®/Qt Quick, a cross-platform application and user interface (UI) framework.

Intel Automotive Services APIs

Intel Automotive Services (IAS) software provides a set of processes that handle media playing, radio, Bluetooth®, hands-free calling, voice recognition, navigation and persistent settings storage. The ICS HMI communicates with these processes via IPC (interprocess communications) interfaces generated from IDL (interface description language) files delivered with the Intel In-Vehicle Solutions SDK (software development kit).

There are a number of advantages to the out-of-process nature of IAS:

- Increased stability as each service can be verified and validated without an HMI
- Increased performance without the need for complex threading considerations
- Asynchronous communication, allowing the user interface (UI) to remain responsive when services are performing blocking operations on vehicle devices

Intel's In-Vehicle Solutions software foundation also provides the cutting edge Wayland window composition system. This allows the HMI to be made of several graphical surfaces, both in and out-of-process, that can be layered and blended together to create a modular and seamless user interface.

Qt/Qt Quick

Qt is a comprehensive C++ application framework that has become the preferred toolkit among embedded touchscreen UI engineers. Qt has a long history of cross-platform C++ GUI development, starting in 1991. Qt is best known for its introspection capabilities and signals / slots object communication system that allow for very loosely coupled and reusable software to be developed. Qt was a pioneer in internationalization, adopting native UNICODE strings from version 1.0 and shipping pre-built translation and localization tools with the framework.

54 B Middlesex Turnpike, Bedford, MA 01730
617-621-0060 www.ics.com info@ics.com

Qt also delivers a declarative user interface programming language called QML and set of components known as Qt Quick. Based on a C++ OpenGL ES 2 scene graph, Qt Quick performs well on embedded systems with graphics acceleration capabilities. QML is a user interface specific language which provides backend data binding, state machines, animations and transitions built into the base language.

IAS — Qt Bridge

ICS provides a library called QtIasBridge that maps IAS calls, responses and events into Qt cross thread signals and slots. This library provides access to the IAS / IPC / API in a manner consistent with Qt's architecture. Since Qt's cross thread signals and slots implement all locking and data marshalling, the HMI implementation does not need to use complex threading paradigms. Using this library, the ICS HMI is able to use IAS's asynchronous threaded design, but avoids all race condition and deadlock situations. The result is increased runtime stability and easier to write and maintain HMI source code.

Design Goals

The ICS HMI system included design goals that provided:

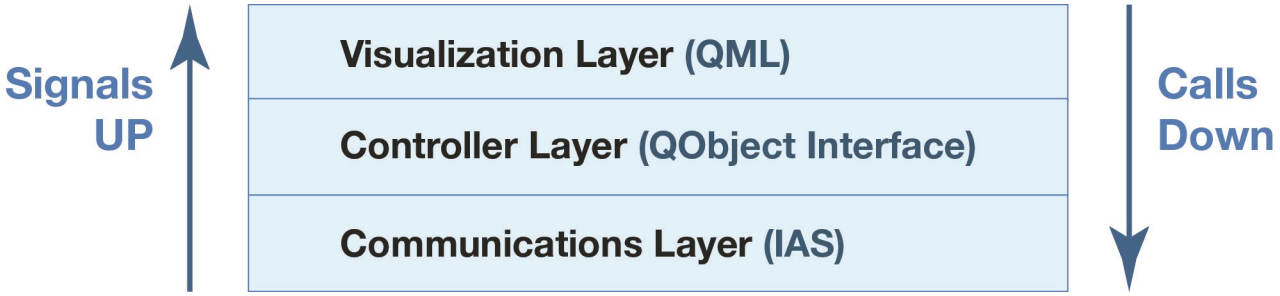
1) a set of C++ classes including all HMI features that can be integrated with vastly different HMI visual designs
2) the ability to test all HMI functionality in isolation without any HMI
3) a set of loosely coupled native applications that can be plugged into the HMI system
4) the ability to deliver new applications without needing to modify, recompile or relink any existing HMI code
5) easy customization of HMI application assets without needing to modify the HMI code
6) the ability for the HMI to be translated and localized for worldwide deployment
7) seamless integration with third-party supplied HMI functionality such as Neusoft® Maps, OpenMobile™ Android Compatibility Layer (ACL)™, Apple® CarPlay and Cinemo™ Media Framework.
8) delivery of < 2 second startup time and velvet smooth runtime performance

These design goals were met using the following design techniques:

Layered Design

The ICS HMI system is comprised of three distinct layers:

The layers on top depend directly on the layer below. As such, they make direct calls to the layer below. The lower layers do not depend on the upper layers and only emit signals to notify the upper layers of state changes. The advantage of this design is the Visualization Layer can be replaced at will without disturbing the rest of the stack.



54 B Middlesex Turnpike, Bedford, MA 01730 617-621-0060 www.ics.com info@ics.com

Ultimately, the features of an HMI are defined by the capabilities of the underlying hardware. These capabilities are exposed to the HMI through a C++ interface from the Control Layer. This decoupling means the controller can be used with a variety of user interfaces. For example, a radio controller that supports the ability to seek, scan or tune to a radio station, save radio presets and retrieve RDS (radio data system) broadcast information can have several user interfaces with an arbitrary look and feel. A secondary advantage is the ability to test the controller C++ code directly, without a user interface at all.

The IAS Communications Layer provides asynchronous access to the vehicle hardware while the Controller Layer caches data from IAS allowing for synchronous access to the current data by the HMI. As such, this allows parts of the UI to be created, recreated or destroyed on demand.

The ability to create parts of the UI as needed allows for minimal startup time. The controller layer architecture allows, for example, the radio controller to start playing the last selected radio station before the Radio HMI (or any HMI) is created. Once the user navigates to the Home Screen or Media only then are those pieces of the UI created on demand. Qt's data binding keeps the UI in perpetual sync with the controller regardless of controller state when the UI is created.

Plug-in Architecture

There are several native applications that combine to create the current ICS HMI. These include:

1) Climate
2) Media (USB, Radio, CD, Video, AUX and Bluetooth AVRCP)
3) Digital TV
4) Hands-free Calling
5) Navigation
6) Vehicle Diagnostics
7) Settings (Some content is injected by other plug-ins for centralized settings)
8) Bluetooth Management
9) Wi-Fi Management
10) Apple CarPlay
11) Android Runtime (Delivers third-party applications)
12) Voice Recognition
13) Weather
14) Home Screen (Displays injected content from other plugins)

Each of these applications are delivered as a directory containing a C++ shared object that provides the Controller Layer classes, a set of language translation files and one or more compiled resource collections that includes a Visualization Layer (QML and assets). These shared objects are loaded by the main HMI (aka View Controller) at runtime allowing for the insertion or deletion of applications without needing to recompile the View Controller. The plug-in shared objects manage loading their own resource and translation files.

Each of these plug-ins provides and registers its own Controller Layer classes and its own Visualization Layer QML with the View Controller. The application plug-in UIs are built with a framework delivered by the View Controller. This framework provides an application shell, dialogs, buttons, sliders and other common components that will be used by all plug-ins for a consistent look and feel. These plug-ins may deliver a variety of user interface implementations to be used with its one set of Controller Layer classes. See the section titled "User Interface Templates."

Surface Composition

Qt's native Wayland support combined with IAS Surface Controller allows the ICS HMI to be comprised of multiple top-level layers blended together via hardware acceleration. The feature allows for modular user interface and integration of out-of-process content.

For example, Cinemo provides high-performance synchronized multi-unit hardware with accelerated video playback. The HMI contains no code to handle playing video, but simply reserves an area of the HMI and configures the Surface Manager to place the Cinemo window at the correct place with the correct size and z-order.

Then the HMI creates a third surface with a higher z-order to be displayed over video effectively making an HMI controlled UI appear blended on top of the Cinemo video window.

View Controller provides a set of QML components that HMI application plug-ins simply use to lay out and configure their external windows appropriately:

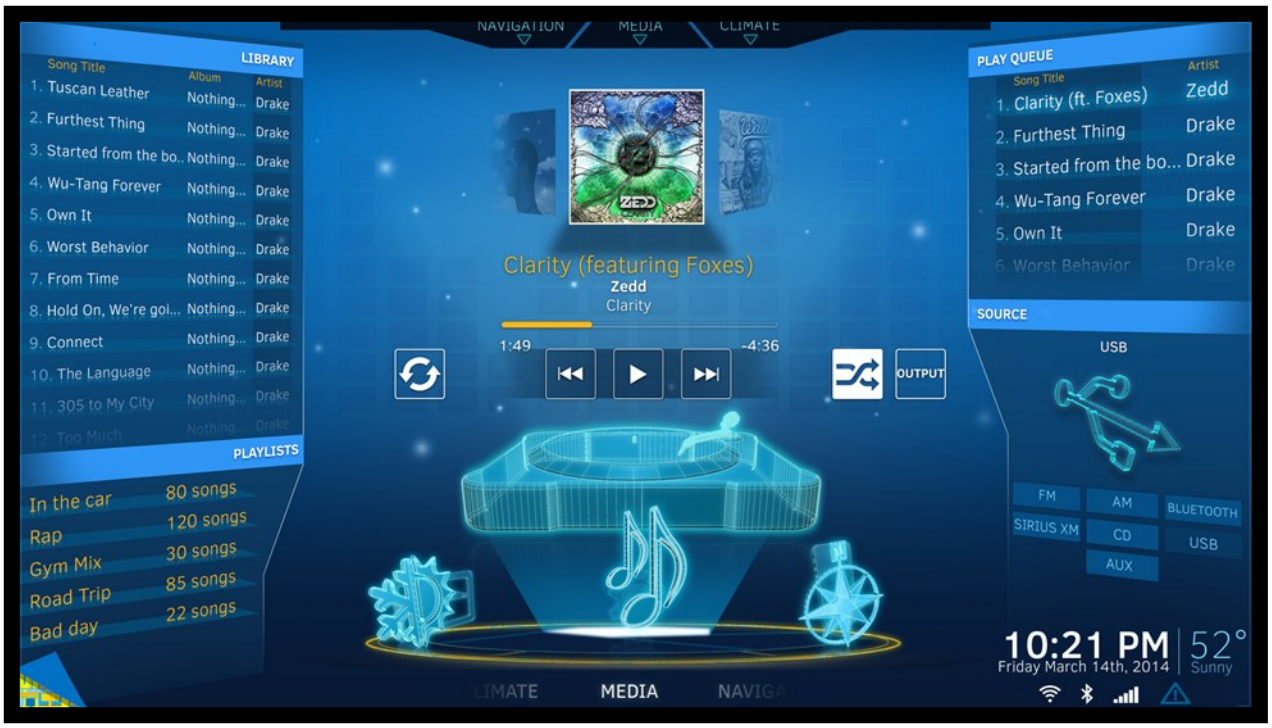
1) Surface: out-of-process windows laid out in app local coordinates
2) Surface: top-level - out-of-process windows laid out in global screen coordinates
3) Overlay: in-process window laid out in app local coordinates
4) Overlay: top-level - in-process window laid out in global screen coordinates

The View Controller via Surface Manager controls the external windows via the Surface and Overlay instances, making sure the external windows are only visible when their corresponding QML item's visible property evaluates to true.

54 B Middlesex Turnpike, Bedford, MA 01730 617-621-0060 www.ics.com info@ics.com

User Interface Templates

The ICS HMI is designed to be UX agnostic. For example, either of the layouts depicted below are deliverable by swapping out the Visualization Layer while using a single Controller Layer. These two templates could be swapped at runtime in under a second without skipping a beat on a playing MP3. Only the latter layout is currently delivered with the ICS HMI.



Phase 1 Template

Phase 2 Template

54 B Middlesex Turnpike, Bedford, MA 01730 617-621-0060 www.ics.com info@ics.com

These wildly different user interfaces are delivered as separate resource collections. Templating could be leveraged to implement a standard full featured HMI, a simple mode HMI and a children's mode HMI deployed as one package. This is an example of a nominal deployment directory structure:

```
/opt/ics/
    vc/
       qqvcon - View Controller Executable (includes a Controller Layer for managing apps)
        vc_phase1.rcc - UI Shell and Framework for applications using the Phase 1 Template
        vc_phase2.rcc - UI Shell and Framework for applications using the Phase 2 Template
    plugins/
        ClimateApp/
            libClimateApp.so - Controller Layer to manage HMI functionality and state of app
            phase1.rcc - QML and assets for Phase 1 Template Visualization Layer
            phase2.rcc - QML and assets for Phase 2 Template Visualization Layer
```

There is no mix and match of templates allowed. The View Controller manages the list of available templates and which template is the current template. The View Controller commands the plug-ins to switch their .rcc file when appropriate and reloads the UI. If an app does not support a template, that app will not re-load as part of the HMI. Since controller state is maintained during the UI swap, the UI loads to the appropriate screen with the appropriate states shown.

RCC (resource compiler .rcc) files are a feature of Qt's Resource System. Inside of the Qt framework the resources appear within a virtual filesystem. The .rcc file is generated via a QRC XML configuration file, but the standard practice is to let the .rcc reflect asset directories from the source tree. Qt internally takes care of memory mapping the .rcc file allowing the OS (operating system) to page the assets efficiently in and out of memory as they are used.

Asset Management

The assets chosen to compose the UI can be customized via Qt's File Selector system. The file selector system combines assets from specially named directories to create asset overlays on top of standard assets. The directories to be selected are configurable by the View Controller and the HMI can use any criteria to determine the list of overlays to use. Any type of file can be embedded into an .rcc. Likewise, any type of file can be overlaid using selectors. Here is an example of color theming assets via file selectors. This is what the virtual directory structure looks like inside of the .rcc files.

```
ViewController
   qml/
        App.qml
        Button.qml
        Palette.qml
        +NightMode/
            Palette.qml
        +GreenTheme/
              Palette.qml
              +NightMode/
                  Palette.qml
 images/
         ButtonPressed.png
             +NightMode/
                  ButtonPressed.png
  ClimateApp/
        qml/
            ClimateAppHmi.qml
   images/
            HeatedSeatOn.png
            +GreenTheme/
                    HeatedSeatOn.png
```

These selectors can be set using QQmlFileSelector::setSelectors(QStringList). Selectors can be nested and there is a left to right order of precedence. Calling setSelectors with a list consisting of "GreenTheme" and "NightMode" will cause this selection criteria to happen:

1) Files from +GreenTheme/+NightMode are preferred
2) If there is no specific +GreenTheme/+NightMode file the +GreenTheme/ file is preferred
3) If there is no +GreenTheme/ file the base level +NightMode/ file is preferred
4) If there is no +NightMode/ asset the base level file is used

Common selectors are color theme, color mode and locale (en_US). Other criteria could also be used such as DPI and screen resolution. These criteria could be read from a settings file or by querying the system.

54 B Middlesex Turnpike, Bedford, MA 01730 617-621-0060 www.ics.com info@ics.com

Make and Model Customization

Make and model customizations are simply implemented as top-level Qt File Selectors. If there are model specific assets, they are preferred. Next make assets are preferred. Finally the rest of the selectors above are implemented. The criteria for choosing the make and model could be a configuration file or, on initial startup, the HMI could query the VIN (vehicle identification number) over CAN (controller area network) and automatically set up the appropriate selectors for make, model and possibly even option packages if that data is available over CAN. That data could be saved to a settings file for quicker access on all future startups.

Here is an example of how the car displayed inside the Vehicle Diagnostics Application is handled for the ACME Corporation line of vehicles:

```
DiagApp/
    qml/
        Car.qml // Usable car for Sedan and SUV
        +ACME/
            +Roadster/
                Car.qml // Roadster needs alterations to UI code
    images/
        CarLeft.png
        CarRight.png
        +ACME/
            +Sedan/
                CarLeft.png
                CarRight.png
            +SUV/
                CarLeft.png
                CarRight.png
            +Roadster/
                  CarLeft.png
                  CarRight.png
```

A template settings file is distributed inside of the resource collection. Among other things, this file dictates which apps are loaded and which color themes are available. Using selectors this configuration can be different per make, model, locale or any combination thereof. Following is an example of overriding the template.json settings file.

```
viewController/
    template.json // Default configuration
    +ACME/
        template.json // Configuration for most ACME vehicles
        +Hybrid/
            template.json // Hybrid has different configuration
```


54 B Middlesex Turnpike, Bedford, MA 01730 617-621-0060 www.ics.com info@ics.com

Screen Orientation

The Qt framework is also used to develop cross-platform mobile phone and tablet apps. Mobile devices often need to reconfigure its layout at runtime due to the user reorienting their screen. Reusing the same HMI code for landscape and portrait configurations is possible using the same technology that Qt provides for that use case. The same software can be used with hardware of varying form factors for different vehicles or option packages.

```
App {
    // UI component definitions and backend bindings
    …
   states: [
        State {
            name: "Portrait"
            when: ViewController.orientation === Qt.PortraitOrientation
           AnchorChanges { … }
           PropertyChanges { … }
           ScriptAction { … }
        },
        State {
            name: "Landscape"
            when: ViewController.orientation === Qt.LandscapeOrientation
           AnchorChanges { … }
           PropertyChanges { … }
           ScriptAction { … }
        }
   ]
}
```

Qt Quick has state machine functionality built into the language and item layout system. Providing more than one layout to a QML screen is as simple as defining a state machine to switch the layout anchors and properties. See the example above.

On occasion assets need to be changed between one orientation and another. Qt's File Selectors can help select different assets for different orientations.

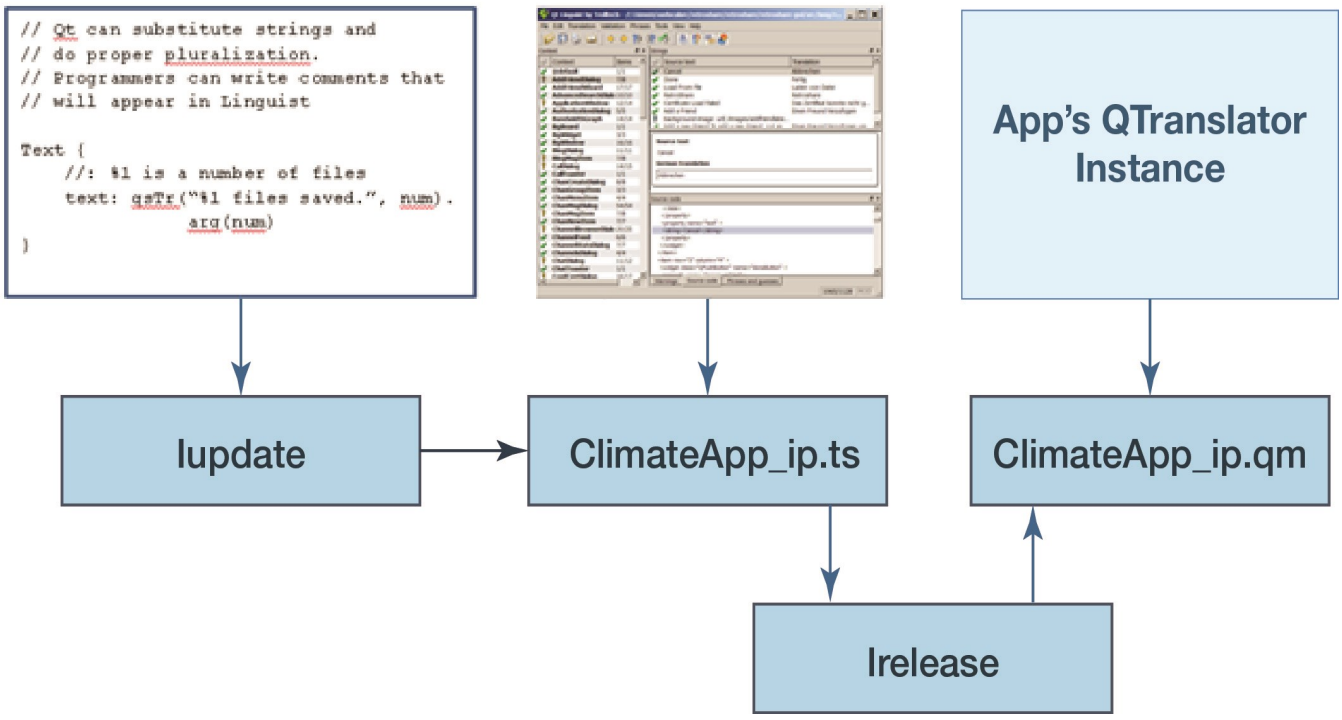
Templates vs. Asset Management

Templates are for creating new, very different HMIs. Asset management via Qt's File Selector is intended to tailor a template to a set of conditions. For example to support the same HMI on different hardware, one could use the resolution "960x540" as a file selector and gain access to pixel perfect assets for that hardware where the base assets are intended for "1920x1080".

If the hardware is so vastly different (e.g., moving from a 19-inch 1920x1080 portrait screen to a 6-inch 640x480 landscape screen) that a new UX is required to be usable, then a new template may need to be implemented to support that use case. Part of the ICS HMI design is the ability to replace the entire Visualization Layer without disturbing the stable and tested Controller Layer.

Internationalization

The developers of Qt, having been primarily located in Oslo, Norway, are acutely aware of the challenges involved in deploying software that fits the user's expectation of native language translation and localization of numbers, dates, times and currency. Qt ships a comprehensive set of localization and internationalization tools. This includes the Linguist suite of language translation tools, QLocale to format text to be displayed and the ability to overlay assets per locale via Qt File Selectors. Below is the translation workflow provided by Qt:



54 B Middlesex Turnpike, Bedford, MA 01730 617-621-0060 www.ics.com info@ics.com

Performance Metrics

The software as implemented per this design loads the HMI to the Disclaimer screen with fully functional climate controls in under 1.5 seconds. Reloading the assets for templating, locale change or theming at runtime takes under one-second while maintaining all state and continuing all HMI activity (radio, video, phone, etc.) during the asset swap and reload.

Conclusion

The concepts used in ICS's proof-of-concept are not limited to IVI applications. They can be used for many applications, especially those that need work on desktop, tablet and handheld platforms. Utilizing the features built into Qt, ICS created a framework that supports different screen sizes and orientations, themes and styles, levels of functionality and locales. In the case of the IVI, the framework allows the same code base to support vehicles with different screen sizes, feature sets and model styles. The result is increased flexibility, a simpler code base and decreased testing needs while delivering applications that blend beauty and simplicity with practical use and technology.

About ICS

Integrated Computer Solutions (ICS) fuses an extensive automotive domain knowledge and engineering expertise with state-of-the-art UX design and graphics capabilities to develop custom, integrated automotive infotainment systems that consumers have come to expect. ICS reduces complexity, costs, redundancies and development time while solving tough engineering challenges up front, increasing the commercial potential of any IVI system. More information about ICS's IVI solutions can be found at www.ics.com/ivi.