Automotive systems are increasingly interacting with the outside world, like smartphones and the cloud, making it imperative to understand how applications are developed, their sustainability, and the difference between developing apps with HTML5 and the traditional native development approach. Managing the interaction between HTML5 and automotive infotainment systems requires technologically advanced approaches in order to yield the best results, especially given the new and complex integration and support challenges associated with HTML5 integration.

### Challenges of integrating HTML5 into automotive systems

Third-party developers may face difficulties when navigating the automotive space. It is safe to say the safety and security of the automotive system is the most important factor when creating in-vehicle applications, since any type of compromise puts an increased danger on the user. Application developers must create apps that restrict usage while driving without interrupting the automotive system. Only specially provisioned applications should have access to the vehicle’s information.

Aside from safety, there are other technical challenges to consider. First and foremost, the technology used in automotive systems must remain relevant and well maintained for the entire lifetime of the vehicle, which averages about 10 years, unlike smartphones with a lifespan of about 2-3 years. It needs to be able to constantly support current technological standards as new developments arise.

There is also the challenge of ensuring the ease of usability in the vehicle. If the vehicle has multiple screens (e.g., head-unit, cluster display, high-mounted display, etc.), the applications must be able to simultaneously run and interact with all of those screens. Furthermore, consumers expect their in-vehicle car apps to resemble those on their smartphones, complete with a rich user interface, simple commands and inputs, and sophisticated animations.

### Benefits of using HTML5 in automotive systems

One of the greatest features of HTML5 applications is they can be easily developed and adapted to function on various mobile devices. This is especially important for integration with automotive systems, as they have their own specific technology-based demands. Today’s market is already filled with plenty of HTML5 apps relevant to automotive systems (think of the plethora of music and GPS apps available), and with the countless developers familiar with and working in HTML5, there will only be more HTML5-based automotive apps in the future.

Also, HTML5 is already equipped with the necessary infrastructure for automotive application development. An added benefit is the style of the apps can be easily changed via CSS (another
critical technology for building Web pages), making this adaptable process a perfect fit for multiple platforms and cloud technologies.

7 steps to building an HTML5 solution

In order to successfully build an HTML5 solution, developers should follow several high level steps:

1. **Build a Browser** - This is the first and most advanced of the 7 steps since it involves licensing or building a browser in-house and integrating it with the target platform. Usually, integration and optimization with both the hardware and input methods are involved while unnecessary components are taken out and performance accelerated. Restricted browser vendors (e.g., Dolphin, Obigo, Access Systems, ZetaKey) can be used. The key advantage of working with a third-party web browser provider is the ability to develop and maintain the solution over the lifespan of the vehicle. Another option is to develop an in-house solution based on the Webkit browser. The Webkit browser can be based on the QT distribution or the latest version can be obtained through the web. The Webkit browser can be further optimized with a faster Javascript engine (e.g., V8 from Google). A final option would be to use another well-maintained open source browser such as Chromium (Google-based).

2. **Native integration** - The browser can be further integrated with native components. The challenging user interface tasks can be handled using native components, either by utilizing browser plugins or by running separate processes, which can interact with the HTML applications via HTML/Socket servers. Examples of these components include a media player, navigation/mapping, hands-free phone integration, etc. Access to this functionality can be given to third party applications though specific APIs.

3. **Create a simulation environment** - To provide third party developers with the ability to create and test their applications, a simulation environment must be provided for each desktop OS (Windows, Mac, and possibly Linux). The simulation environment should directly reflect the target platform browser environment, incorporate a user interface to allow hardware buttons to be simulated, be able to install and remove applications, and provide a console output and web application debugging capabilities.

4. **Hardware button integration** - There are typically a variety of controls such as touch screen, steering wheel buttons, rotary knobs, etc., in automotive systems, and they must be integrated within the platform. It is recommended that these controls not require a special API, and be developed as standard navigation controls, (e.g., a rotary button that acts as a tab-control which changes the active focus). However, these controls can expose an extended JavaScript API to allow certain applications to direct them differently. The browser can inspect if the active element has a registered handler for the special control and if so, utilize its function. If there is no registered handler, the browser can resort to the standard behavior.

5. **Application JavaScript Architecture** - Ability to run third party applications is one of the major requirements and pros of an HTML5 based automotive system. The system must provide a platform where the applications can be downloaded, executed, and can communicate fully with one another. A possible architecture is drawn out and described in detail in the JavaScript section below.

6. **Proprietary SDK** - Additional options such as access to vehicle information, GPS, diagnostics information, driving status, etc. are fairly common in automotive systems today. To provide an improved integrated experience, the system may also provide a proprietary software development
kit (SDK) to allow third party applications to communicate with each other. This can be achieved by using a JavaScript API that applications are able to easily link to and use. It is suggested that a standard JavaScript API be used for standard functionality such as location API (GPS) because it will allow the applications to be more portable and easier to develop.

7. **Multiple screen integration** - Applications in the automotive environment can operate on multiple screens, such as a center stack display where the main application can run, and other displays such as the instrument cluster where additional information can be displayed (e.g., Internet radio station, the next turn-by-turn instruction, etc.). A possible multi-screen architecture is illustrated and described in the multiple screen architecture section below.

**Application JavaScript architecture example**

When working with JavaScript architecture, HTML5 is implemented as the primary user interface framework using the standard HTML approach. Abstractions are created through the JavaScript libraries, and independent JavaScript components and applications are used for a modular approach. For more intricate user interface elements, the browser can extend with native components. Modules are kept separated for easier development and application maintenance. jQuery is used as the base user interface framework because of the availability of third party user interface widgets. User interface frameworks are also available as an extension of jQuery.

The lifetime and window management of the applications are managed through the App Management Framework, and these applications run separately through `div` elements (used for generic organizational or stylistic applications) and `iframe` elements (inline floating frames).
App Connector Library is the communication liaison between the applications and the App Framework Management, which are separate modules. Both the App Management Framework and the App Connector Library utilize JavaScript.

The Extension JS Library is an optional feature. Given that everything performs properly using pure JS/HTML, this feature is unnecessary. Its role is to expose the native plug-in functionality as JavaScript.

A multiple screen architecture example

As seen in the illustration above, the automotive system can run multiple applications by using either a high mounted display (HMD), an instrument cluster display (ICD), or a center stack display (CSD). The applications communicate with the Intra-System Messaging Module (ISM) via an AppHost (Qt). The ISM allows applications to broadcast messages or register message listeners. The AppHost broadcasts messages to other hosts and in return these are translated to JavaScript and are broadcast back to the ISM. Animation or other UI synchronization between the screens can also use this mechanism.

The benefits of HTML5 far surpass its challenges

Even though the primary challenge to successfully creating an HTML5-based system is building and maintaining an optimized browser, doing so specifically for automotive systems will allow the vehicle to remain relevant for its entire lifetime. However, there are other major automotive-specific factors that must be taken into consideration, including vehicle-specific Human Machine Interfaces and multi-screen systems. The high-level architectures presented above provide an effective approach for addressing some of these unique automotive difficulties, opening up mass potential for HTML5 in the automotive world.

Also see:

- In-vehicle product differentiation: open standards vs open source
- Understanding Mobile Apps for Automotive
- Open Standards and Product Differentiation
- Mobile OS Architecture Trends
- HTML5 for automotive infotainment: What, why, and how?
• Graphics Acceleration for HTML5 and JavaScript Engine JIT Optimization for Mobile Devices