Radioplayer

An Analysis of support for DAB Digital Radio and Hybrid Radio in Android Automotive

June 2021

1. Executive Summary

Radioplayer is the international broadcaster-backed organisation working with car manufacturers (OEM Original Equipment Manufacturer) to create brilliant hybrid radio user experiences with broadcast radio at their heart. Radioplayer has been working with Google's Android Automotive OS since 2019 and has developed a hybrid radio app (DAB and FM working with IP) that is available for OEMs to use and adapt as their own radio experience. Our aim is to ensure hybrid radio, powered by free-to-air broadcast, is the standard radio experience described in Android Automotive. Our priority is the OEM broadcast radio experience via the "Radio button", known as a system app, where radio functionality is accessed through the Hardware Abstraction Layer (HAL).

In discussion with Google, our feedback was requested and this report summarises where Radioplayer believes Android Automotive should improve the DAB broadcast experience and the requirements for hybrid radio. This has also been shared with other industry stakeholders working in this area such as WorldDAB and NAB Pilot.

We have also consulted with OEMs, some of whom have provided feedback which is shown throughout the report. We have incorporated comments in to the report and also shown some of the comments separately to provide additional context. Some of the questions raised here will require further discussion between broadcasters and OEMs. The analysis is grounded in the user experience (UX) of radio in the car and Radioplayer is a member of the WorldDAB automotive user experience group who have developed some DAB UX guidelines whose use cases are referenced in this report.

Android Automotive has evolved significantly since the initial release, and some DAB specific knowledge has evidently been provided to the developers at Google. As a result, the majority of expected DAB functionality is available to OEMs to develop radio apps with. That which is missing prevents providing the best user experience of DAB and hybrid radio in the car, potentially a regression from existing line-fit DAB radios. If those deficiencies are addressed in Android Automotive, an OEM should be able to produce a user experience comparable with the best radio (streaming) apps.

We have also considered the situation for other apps outside of OEM system apps, known as second or third-party apps, who could access radio functionality via the Android Automotive Media Browser in the future. This is not a priority and is not included here, but rather a potential later phase of development that requires further discussion.

Key Recommendations for Android Automotive OS:

- DAB Service and Programme Information (DAB SPI) is the way in which broadcasters deliver station logos and other programme information such as daily broadcast schedules with detailed information about every programme event, start and end times, event names, genre and descriptions. There must be a pathway through Android Automotive for DAB SPI to pass this data to the radio app.
- To enable a good hybrid radio experience, better support is required to help determine signal quality and manage switching between broadcast and IP streaming, and methods provided to assist with

seamless switching are required. Defining a standard signal "quality" value would need to include measures such as reception quality, audio quality, RDS reception quality, and potential for other broadcast radio sources through linking.

- The implementation of DAB Service Linking between DAB-DAB, DAB-FM (hard and soft) is critical to a good user experience, and better support for switching is required within the HAL for system apps.

Key recommendations for OEM system developers:

- It should be mandatory for the tuner driver to be able to receive and decode DAB SPI for the HAL.

2. Introduction

Android Automotive is a set of automotive specific extensions to the Android Open Source Project (AOSP). These extensions are to handle use cases and hardware that are (largely) unique to the automotive environment.

Broadcast Radio is one of the provided extensions, intending to provide an interface into radio hardware for AM, FM, HD and DAB. Whilst (crude) radio functionality has been implemented in previous Android devices (notably mid-range Android smart phones), it is now standardised within Android. The extensions have been designed and coded by developers at Google, and the code shows signs that at least some of it has been informed by radio tuner providers.

Android has a layered architecture, and code has to be provided to bridge functionality between each layer. If a bridge is missing or blocked between two layers, access to that functionality is lost to end applications. Similarly, if compromises are made translating radio functionality into Android's standard model(s) for functionality, the functionality can become ineffectual.

This report assesses how well Android Automotive provides functionality for DAB and for hybrid radio for the core radio experience provided by the car manufacturer and installed at manufacture, known as a "system app".

The functionality of line-fit radio receivers in vehicles has grown in complexity with the introduction of RDS, then DAB and now hybrid radio. System apps must be capable of achieving at least the same level of functional competency as the best line-fit radios. This analysis includes some references to FM and hybrid radio functionality, reflecting that best-practice is to create a harmonised experience of radio, regardless of source.

The WorldDAB Automotive Group User Experience Working Group has defined a baseline for user experience of broadcast radio in cars, and apps must also be capable of supporting that user experience as a minimum.

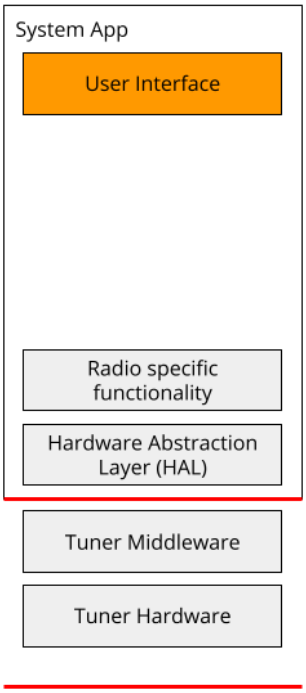
These two areas (functional competency and user experience) form the benchmarks for this analysis of capability.

The relevant specifications for broadcast radio (RDS, DAB, Hybrid Radio) allow for a range of functionality that is impossible to completely implement, given the extent and permutations of functionality possible. In this analysis, we only consider the functionality required to support the use cases.

3. Constraints on Access to Functionality

Apps provided by the vehicle manufacturer are considered privileged system apps. These apps are able to access hardware directly. As such they can implement the entire broadcast radio functionality, from the user interface to controlling the hardware, in one closed stack.

OEM comment:
[The exact architecture] is dependent on the hardware/software and the partitioning in the system



a. Tuner Middleware

There is a layer of software below the Hardware Adaption Layer (HAL) which is developed by the OEM/Tier 1 and in turn communicates with tuner API and tuner driver layer. This converts functionality described by the APIs in the HAL into instructions to the silicon chip. In the case of DAB, almost all of the DAB specific functionality is handled in this layer, for example:

- Decoding of the FIC (Fast Information Channel), and the FIGs (Fast Information Groups) to create a list of the stations on an ensemble, their SIds, their names and their audio configuration
- Decoding of DAB SPI to acquire the station logos, and programme information
- Decoding of DLS Text (and/or DL+) from the audio stream
- Decoding of DAB Slideshow to acquire visuals and display them to the user at the correct time
- Routing the audio signal from the silicon chip directly to the car audio system
- Managing the switching between DAB-DAB and DAB-FM for hard-linking
- Delaying audio to ensure that switching between DAB and FM is seamless
- Decoding of announcements

This code represents the vast majority of “DAB specific” knowledge required to make a radio work, and it functions entirely out of sight of Android, leaving Android only having to deal with outcomes / simple controls. In most cases, this is helpful, but in some cases this inaccessibility prevents functionality working correctly. For example, hard-linking is handled invisibly in this layer, but soft-linking must be handled in the application layer (including switching to streaming), and there is no way to access or control signal quality assumptions or audio streams.

b. Implementing Hybrid Radio Functionality

Hybrid Radio uses IP connectivity to enhance broadcast radio. In most use cases, that additional IP delivered functionality is best implemented by integrating standard/open libraries.

Better integration with the tuner hardware is required is to support switching between broadcast radio and the same audio as an IP stream. In this case, a more consistent understanding of "signal quality" and the ability to control both the IP and broadcast radio streams to allow for seamless blending is required. This is something we want to define in more detail in discussion with OEMs and Google.

The current method of identifying radio services is insufficiently precise for hybrid radio functionality. For example, the DAB_SID_EXT property does not include the current Ensemble ID. Some vendors provide proprietary extensions to overcome this shortcoming, but they should be standardised.

OEM comment:
Currently this can only be done by extensions of the broadcast radio HAL

4. Common Functionality in Radio Apps

Listeners can choose from a wide variety of streaming apps to listen to the radio with eg broadcaster apps, and this creates an expectation of user experience that should be replicable in the stock Android Automotive radio experience. Broadcasters encourage use of their own apps, which means they're likely to make them compatible with Android Automotive (technically and usability), and promote their installation to drivers.

Some of the common functional elements of these apps are:

- Rich visual design that makes most use of the graphics capability of the device
- Clarity of navigation
- Fluid and responsive to user action
- Iconography and imagery
- Easy to move between listen live and listen on demand
- Pause / resume
- Access to more detailed content
- Personalisation / Recommendations

5. Use Case Analysis

These use cases are drawn from the WorldDAB Automotive Group User Experience Working Group Guidelines. Each use case and its primary requirement(s) is/are listed, along with the extent to which the requirement(s) can be met by system apps interfacing directly to the tuner hardware – **HAL Support**

Use Case	Functionality Required	HAL Support
"Selecting RADIO starts playback of the previously used station"		
	Last tuned station is held persistently across app sessions	Yes
" Present an integrated and unified A-Z list of all available stations (FM, DAB, IP) which defaults to DAB or present an A-Z list of all available DAB stations"		
	Collate a list of all the receivable stations	Yes Note: Currently the HAL only considers a subset of DAB Band III frequencies
	Acquire the names of those stations	Yes Note: 8 character station names are supported, but use should be deprecated
	Acquire the logos of those stations from DAB SPI (Over The Air)	Partial Tuner Driver must implement a DAB SPI decoder. Only one non-specific size logo can be provided for each station, but the standard supports multiple logos. Discussion is needed to decide if: all provided logos should be acquired and made available to apps, or; the app should register for a subset of logos to be acquired, or the OEM decides on a subset of the standardised logos sizes. **OEM comment:** To save space and only save logos that are needed it is best to select one best fitting size logo. To deliver all the possible logos is too big a load on the memory if they're not being used by the OEM app
	Merge lists across FM/DAB and de-duplicate	Partial RDS PI codes and DAB Sid codes are provided. Hard Link sets may not be provided, so complete de-duplication isn't possible, unless it's handled in the Tuner Driver. Further discussion is required to decide if Hard Link information should be handled entirely by the Tuner Layer (and so invisible to apps), or provide a method for apps to be aware that services have been de-duplicated based on Hard Link set membership. The behaviour when Hard Link

	settings change and the effect on service de-duplication needs to be standardised. **OEM comment:** It is best to keep the de-duplication handled in the Tuner Driver
Display the list of services	Yes
Display / sort by genre	Yes PTY values are provided
Allow “next” and “previous” navigation of stations once tuned in	Yes

“**In a Hybrid Radio, the best platform signal will be selected”**	
If there are no “good quality” broadcast signals, switch to streaming	No Opaque value of “quality” makes it impossible to decide when to switch. Defining a standard “quality” value would need to include measures such as reception quality, audio quality, RDS reception quality, potential for other broadcast radio sources through linking. Any switching decision should take into account broadcast alternatives prior to switching to IP. **OEM comment:** A generic standard definition of quality would be interesting to define. The quality measure must include: Bad reception, audio reception quality, quality of RDS decoding possibilities, hand-over preparations, quality of alternatives, good reception, etc…
If there are “good quality” broadcast signals, switch back to the “best” signal	No Opaque value of “quality” makes it impossible to decide when to switch **OEM comment:** The concept is dependent on the OEM preference, therefore all information needs to be available on the interface in order for each OEM to decide on the strategy.
Make the switch between broadcast and IP seamless	No DAB (or FM) Audio is not available as a stream within Android, so impossible to buffer/time align with IP at an app level. **OEM comment:** It would be very helpful to provide an interface to provide the audio data further up the layers in order to be able to implement seamless

	switching to IP. **OEM comment:** Seamless blending should all be done below the Broadcast audio HAL because it requires realtime audio handling

"The station list should be automatically updated when new stations become available or unavailable"	
The station list is updated if station availability changes	Yes
The updated list is shown to the listener	Yes

"The action to set a DAB pre-set should always be to press and hold a button when listening to the wanted station"	
The current station is saved along with all the potential reception methods	Partial RDS PI codes and DAB Sid codes can be saved on a preset. Hard Link sets may not be provided, not all possibilities can be saved. This puts the onus on the tuner firmware to find alternative services to the one selected by the user, which may not be supported by all hardware. **OEM comment:** It is expected that the tuner will find alternative in case a preset is recalled, therefore no need to save Hard-Link sets.

"For FM/DAB tuners, the radio should search for the best signal on FM, DAB, DAB-DAB, for the station being listened to. This happens automatically."	
Maintain a list of ways to receive the current station and use the "best" one	Partial Decision on "best" signal is made by the Tuner Driver and is invisible to Android Note: Not clear if FIG0/6 inhibiting of DAB-FM hard-links is implemented
If the "best" signal changes, change to another signal	Partial Decision on "best" signal is made by the Tuner Driver and is invisible to Android
The listener can disable this feature if they want to	Yes It's possible to disable DAB-DAB and DAB-FM independently. Soft-linking and hard-linking must be individually configurable.

"**[Listeners want] information about what they're listening (eg; station name, station logo, now playing, programme name) clearly presented on the screen**"	
Display the current (optional: and next) programme name	No Tuner Driver must implement a DAB SPI decoder. There are no metadata containers for "programme" information (as opposed to ProgramInfo which holds information about the current station and current programme event)
Further programme information is accessible	No Tuner Driver must implement a DAB SPI decoder. There are no metadata containers for "programme" information (as opposed to ProgramInfo which holds information about the current station and current programme event)
The station programme schedule is visible / browsable	No Tuner Driver must implement a DAB SPI decoder. There are no metadata containers for "programme" information (as opposed to ProgramInfo which holds information about the current station and current programme event)

"**Always show the latest text message [from the currently selected station]. Show the station logo until the first visual is received, then show it automatically.**"	
Acquire and display the text message	Yes Tuner Driver must map DAB DLS into RDS_RT metadata key. This mapping should be explicitly defined.
Update the screen as soon as the text message changes	Yes
Remove the text message immediately if requested by the broadcaster	Undefined The Tuner Driver needs to signal a CLEAR message as a null RDS_RT
Display Artist / Title separately (DL+ support)	Partial Only Artist and Title are defined, but not other DL+ fields
Acquire and display the station logo [at a good quality resolution]	Partial Tuner Driver must implement a DAB SPI decoder. Only one non-specific size logo can be provided for each station, but the standard supports multiple. As for station logos for navigation discussion is needed to decide if: all provided logos should be acquired and made available to apps, or; the app should register for a subset of logos to be

	acquired, or the OEM decides on a subset of the standardised logos sizes. **OEM comment:** It should be possible to define over the API, or per Coding which is the preferred resolution to provide all this data. A clear mapping of the Metadata_Key must be provided.
Acquire and display a visual	Yes Tuner Driver must implement a DAB Slideshow decoder. Tuner Driver must handle caching and TriggerTime correctly
Update the screen as soon as the visual changes	Yes Tuner Driver must implement a DAB Slideshow decoder. Tuner Driver must handle caching and TriggerTime correctly
Display the best resolution visual available	Yes NOTE: There may be an assumption this is “small” 200x200px images

“**The default settings for the DAB radio [should] provide the best user experience** “	
Alphabetical station listing	Yes
Automatic display of text and visuals	Yes
Automatic switching to “best” quality signal	Yes

ANNOUNCEMENTS: Relevant and helpful announcements should be given (interruptive) priority

OEM comment:
Currently all announcements can only be supported through extensions.

OEM comment:
An extension to define all the possible announcements and their status (on/off) would cover this topic – nevertheless, the API currently does not cover it yet.

Configure which announcements the listeners wants	Partial Only supports: Emergency, Event, News, Sport, Traffic, Warning, Weather
Monitor (the available) services for announcements	Yes Tuner Driver must implement announcement masks on DAB services
Notify / interrupt listening when the start	Yes
Resume normal listening when they end	Yes

Display to the user that announcements are provided	Partial Only supports: Traffic (e.g. RDS TP)

6. Use Case Gap Analysis

In most cases, the required functionality is available to system apps interfacing to the HAL layer. Issues which may need addressing are as follows. We have listed these in order of priority for broadcasters:

a. DAB SPI – Station Logos and Programme Information

The model assumes that the Tuner Driver will implement a DAB SPI decoder to be able to present a station logo to the application layer. DAB SPI can also contain programme schedule information for radio stations.

There is no guidance on how to implement this DAB SPI decoder. That probably falls outside the scope of the Android project, but there may need to be guidance on how to implement DAB SPI decoding to meet the use case requirements.

DAB SPI can provide logos in different styles, resolutions and aspect ratios, which can be used at different places in the UX design. For example, a station can provide a square icon of its station logo to be used in navigation lists and on presets, but a full version of their logo in a high resolution rectangular format for full screen display during playback.

The current Android model assumes one logo (of indeterminate size) which is simply resized to fit the space available, which will often result in poor quality images if they are significantly resized from their native resolution. There is no guidance on which logo the Tuner Driver should select from those transmitted over DAB SPI, which will lead to inconsistency and confusion.

Programme Schedule Information is available in DAB SPI, but there is currently no mechanism to represent this in the HAL. This means it's impossible for a system app to show the current and future programme information.

Recommendations:

- Guidance is provided on how to implement a DAB SPI decoder when integrated into Android Automotive.
- The version of the station logo which is considered to be the "primary" logo (that is – mapped to the existing station logo object) is standardised
- Clarify which Media Metadata key represents the station logo.
- Objects are defined to hold all versions of a station logo in the HAL
- Objects are defined to hold the Programme Schedule Information in the HAL

b. Hybrid Radio Switching

Basic hybrid radio switching requires a (reasonably) standardised representation of "good" quality reception and other standardised indicators (such as "mute" status for digital radio, or an explicit "switch" flag), so that the system app can stop playing DAB radio and start playing a stream (or vice versa). In this case, there is no smooth transition between the sources.

If a smooth transition is required, there must be time alignment between broadcast and the IP stream. Whilst various methods exist to achieve this, they all require at least for the broadcast audio to be "time-stretched" (as it is for DAB-FM switching) to time align with the IP stream. The amount of this timing skew can be

calculated by analysis of two streams in the app or within the radio tuner hardware. In many cases, the radio tuner hardware supports this time alignment functionality natively, as part of the requirement to time align DAB-DAB, FM-DAB, HD-FM signals.

Recommendations:

- There should be standardisation of the values of "signal quality". Defining a standard "quality" value would need to include measures such as reception quality, audio quality, RDS reception quality, potential for other broadcast radio sources through linking.
- Broadcast radio audio from the Radio Tuner is available as a digital audio stream within the Android media framework, so it can be handled synonymously with an IP acquired stream. This would allow a system-app to do time alignment, switch and playback of audio, rather than relying on the Radio Tuner to do so.

c. Service Following (Linking Information)

The model allows the tuner to indicate "secondary identifiers" for each service. It isn't clearly defined if the secondary identifiers relate to alternative sources of the identical service, or can also include "related sources" ("soft-links").

The association between bearers can be implied where Sid and PI codes are the same (although not in all cases), but it's also possible for the broadcaster to declare additional relationships explicitly through service linking information.

Whilst the Tuner Layer appears to take responsibility for following hard links automatically, and just notifying the system app of the change, it doesn't do the same for soft links (related services). Soft link information is available at the HAL Layer.

Critical in the process of service following are the decision points to trigger switching of source. If the Tuner Driver handles Hard Links, it will do so without the system app knowing exactly why it's changed. Whilst there's a Signal Quality property available to the HAL, there is no standardisation of what constitutes "poor quality", which would trigger switching.

Also to note, there is a statement in the documentation "For example, RDS PI code, which may be translated to the call sign in the US.". This isn't true in the US, or elsewhere.

Recommendations:

- Clarity is provided on how hard links and soft links should be translated into primary and secondary identifiers for services
- Guidance is given on exactly the behaviour expected of each linking strategy. Is the Tuner Driver expected to follow hard links and notify the app, but provide soft link information to the app for it to manage service following?
- The Tuner Driver should adhere to any DAB-FM implied linking inhibited by FIG0/6

d. Announcements

A system app can ask for notifications of a subset of DAB announcements. The Tuner Driver must implement the detail of deciding which announcements to forward to the HAL, and that leaves open potential confusion about announcements only applying to the current tuned service, or if announcements from any service on the currently tuned ensemble(s) should trigger a callback.

Recommendations:

- Clarity is provided on how the Tuner Driver should respond to announcements signalled on the currently tuned ensemble(s)
- Extend the defined announcements to cover all standardised announcement types

OEM comment: Extend the number of supported announcements. Currently supported: EMERGENCY, WARNING, TRAFFIC, WEATHER, NEWS, EVENT, SPORT, MISC (for others)