

androidauto

Head Unit Integration Guide

Version 1.3.0

Last Updated: 2016-08-10

Google Confidential & Proprietary



Table of Contents

Introduction

About Android Auto Projection (AAP)

Channels

Mobile Device

Head Unit

Conventions

Head Unit Requirements

Screen Support

AAP Integration

Receiver Library and Protocol Versions

AAP Receiver Library

OS Adaptation Layer

Software Requirements

AAP Receiver Interfaces

Version Compatibility

Establishing AAP Connections

Overview

Connect AAP

Service Discovery

Manage Applications

Connect Bluetooth

Grant Video Focus to MD on request

Run Tutorial

Run Disclaimers

Start Android Auto UI

Setting Up Transport

Universal Serial Bus (USB)

Configuring AOAP

USB Mode Selection

Head Unit Responsibilities

Mobile Device Responsibilities

USB Charging

Authenticating

Authentication Certificates

Authentication Sequence

Setting Up Bluetooth

Bluetooth Pairing & Connection

BluetoothService Announcement

Protocol Messages for Bluetooth
Pairing

HFP Connection

Additional Bluetooth Profiles

Bluetooth Reconnection

Integrating Video

Selecting Video Resolution

Request the Highest Resolution Possible

Selecting a Density to Report

Matching Physical Display Size

Required Video Stream Support

Video Focus

Projection Mode

Native Mode

Overlay

Mode Examples

Video Focus Requests

Latency, Flow Control & Clock Skew

Integrating Audio

Audio Streams

- UI (device-to-vehicle)
- Guidance (device-to-vehicle)
- Voice (device-to-vehicle)
- Media (device-to-vehicle)
- Microphone (vehicle-to-device)
- Legacy In-Call (bi-directional)

Audio Focus

- Audio Focus Requests
- Audio Focus Notifications
- Audio Focus Initialization
- Audio Focus Policy
- Audio Focus Priorities
 - Contention & Media Mixing
- ASR Focus
- Navigation Focus
- Media (Music) Focus
- System Stream Support

Audio Mixing Guidelines

- General Audio Notification Guidelines
- For Mixing Streams
- For Non-Mixed Streams
- For Audio Focus & Phone Calls

Audio Focus Examples

- Audio Focus Mixing vs. Non-Mixing
 - Native Nav + MD Music
 - MD Nav & Music + Native FM Radio
 - MD Music + MD Phone Call

Latency, Flow Control & Clock Skew

Volume

Integrating Input Devices

Touchscreen

Buttons

- Button Key Events
- Implementing Button Interface

Rotational Controller Devices

- Dial Key Codes
- Implementing Device Interface

Touchpad

- Implementing Device Interface

Integrating ASR

Initiating Automatic Speech Recognition

- Short-Press
- Long-Press
- Voice Recognition Session
- Cancelling or Restarting a VR Session

Implementing Voice Recognition (VR) Actions

- Short-Press Standard Session
- Short-Press Restart
- Short-Press Cancel
- Long-Press Standard Session
- Long-Press Restart
- Long-Press Cancel

Speech Audio Format

- Processing Requirements
- Processing Recommendations

Integrating Vehicle Data

Driving Status

Fuel

Navigation

Privacy

Safety

Other

Integrating Navigation

Navigation Focus Request

Navigation Focus Notification

Policy

Integrating Application Status & Data

Navigation Data

Navigation Subscription

Navigation Status Event

Next Turn Event

Next Turn Distance Event

Next Turn Enum

Navigation Data Example

Accessing Media Data

Media Playback Status

Integrating Vendor Extension Channels

When to Use VEC

Architecture

Android Mobile Device

Using Verification Certificates

Sample Applications

Head Unit

Sample Implementation

Use Cases

Autostart Application

Determine Android Auto
Compatibility

Initialize Vendor Extension Channel

Frequently Asked Questions

Integrating Radio

Radio Service Discovery

Radio Properties

Messaging Between HU and MD

Example Radio Sequences

Radio Playing During MD Connection

Tuning to Station in Projection Mode

Tuning to Station in Native Mode

Scanning Stations in Projection Mode

Audio Focus When Integrating Radio

Radio Hard Buttons

Appendix A: Vehicle ID

Unique ID per HU

Minimum 64 Bits

Example: Generate Vehicle ID

Appendix B: Document History

Introduction

Today's mobile devices are packed with technology—the latest hardware and software, high-speed mobile networks, and an ever-expanding universe of connected services and applications. Their convergence empowers users with new ways to communicate, enjoy digital content, access information, learn, and navigate the world around them. Unfortunately these devices were not designed to be used while driving.

Android Auto (www.android.com/auto) enables seamless integration between mobile devices and vehicles, allowing drivers to access and control technology on mobile devices using an interface that's easier and safer to use while driving. This document describes the integration that enables automotive in-vehicle infotainment (IVI) systems to leverage the compute power, connectivity, and applications of modern mobile devices through Android Auto Projection (AAP) technology. It is intended for automotive infotainment engineering development teams who want to integrate AAP on a head unit. It does not describe the low-level AAP protocol implementation abstracted by a sender and receiver library on both the head unit (HU) and the mobile device (MD). For details, refer to the [AAP receiver library documentation](#).

About Android Auto Projection (AAP)

AAP bridges mobile devices and vehicles to enable drivers to access the capabilities of the device through the physical human machine interface (HMI) controls provided by the vehicle. The mobile device manages all user interface (UI), software logic, connectivity, and compute power for applications, and projects these applications into the vehicle through AAP.

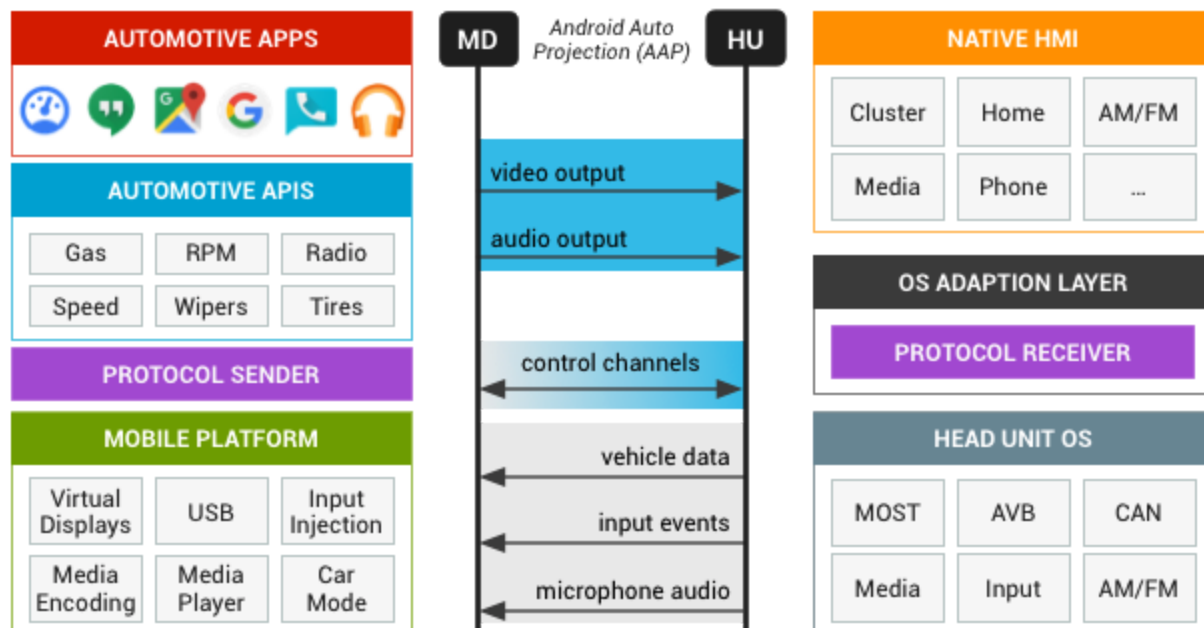


Figure 1. Android Auto Projection (AAP) architecture.

AAP architecture leverages the strongest attributes of both vehicle and mobile device to create an optimal driving experience.

From the **vehicle**, the Android Auto integration leverages:

- Human machine interface (HMI) better suited for use while driving
- Highly accurate vehicle data with better information about the driving environment

From the **mobile device**, the Android Auto integration leverages:

- Latest computing hardware platform
- Latest software platform
- Latest high speed data connectivity to the cloud via latest mobile data networks
- Evolution of hardware, software, and connectivity at consumer electronics pace

Channels

The platform implementation supports USB 2.0, but the protocol itself is transport-agnostic and runs over any transport with sufficient bandwidth (future implementations may utilize Wi-Fi Direct). The AAP protocol includes separate channels to manage data streams between mobile device and vehicle:

- **Control** (*bi-directional*). Manages link initialization and setup of channels for media, input, etc.
- **Video Output** (*device-to-vehicle*). Sends H.264 video from the mobile device to the vehicle for display on the main console display.
- **Audio Output** (*device-to-vehicle*). Carries audio from the mobile device to the vehicle for output through the vehicle speakers (AAC with 48k, AAC or PCM for 16k).
- **Vehicle Data** (*vehicle-to-device*). Carries vehicle-associated (GPS, wheel speed, etc.) data from the vehicle to the mobile device.
- **Input** (*vehicle-to-device*). Sends input events to the mobile device from input devices on the vehicle, such as touchscreens, buttons, controllers, etc.
- **Microphone Audio** (*vehicle-to-device*). Used by the mobile device to receive audio captured by the vehicle microphone.
- **Bluetooth** (*bi-directional*), **Navigation Status** (*device-to-vehicle*), **Media Browse** (*device-to-vehicle*). Used for communication of Bluetooth data, Navigation status and information, and Media browsing information, respectively.

Mobile Device

The architecture on the mobile device consists of the following components:

- **Mobile Platform.** Provides the operating system and application framework that powers the mobile device.
- **Protocol Sender.** Provides the mobile device AAP implementation of and plugs into mobile platform interfaces (video, audio, input, etc.) for a seamless user experience.
- **Automotive APIs.** Operates on top of the protocol sender and defines a standardized API for developers to create applications using the head unit to interact with a driver.

Throughout this document and its diagrams, the mobile device is referred to as the *MD*.

Head Unit

The architecture on the head unit consists of the following high-level components.

NOTE

This document describes high-level components unique to AAP and does not provide a detailed overview of the entire head unit software stack.

- **Head Unit OS.** The operating system (Android, Linux, QNX, Windows, etc.) environment software stack that powers the on-board infotainment experience.
- **OS Adaptation Layer.** The OS-specific abstraction layer between the head unit OS and the AAP receiver. The layer implements AAP receiver bindings and translates the message call and data to native head unit OS subsystems. Google provides reference implementations for Android and QNX in source form, but does not provide a commercial implementation.
- **Android Auto Projection Receiver.** The head unit implementation of AAP that connects with unit OS interfaces (video, audio, connectivity, input, etc.) through the OS Adaptation Layer to enable an integrated user experience. Google provides the receiver library as portable C++ source code.

Throughout this document and its diagrams, the head unit is referred to as the *HU*.

Conventions

The use of “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” is per the IETF standard defined in [RFC2119](#).

Head Unit Requirements

To support AAP, automotive head units (HUs) MUST include the following minimum capabilities (unless explicitly specified otherwise):

Component	Requirement	Comments
<i>Connectivity</i>	USB 2.0 Hi-Speed Host - Standard USB-A port >50 Mbps/s throughput - Support USB 2.0 CDP specification - Host support for Android Open Accessory (AOA) protocol - Support USB Hub (may be exposed in HU debug mode only) Bluetooth - Hands-Free Profile (HFP) 1.5 - PIN or numeric comparison pairing methods	The HU SHOULD equip USB current-limiting and adequate power supply to avoid VBUS shutdown¹. Cellular data is OPTIONAL.
<i>Video</i>	H.264/AVC Baseline Profile hardware decoder: - H.264 BP level 3.1 REQUIRED for both sides to guarantee 800x480 - H.264 BP level 3.2 REQUIRED for 720p (if HU supports this resolution) - H.264 BP level 4.2 REQUIRED for 1080p (if HU supports this resolution) - Max frame rate of 30 FPS or 60 FPS - Frame rate (continuous or noncontinuous) between the max frame rate and 5 FPS - Any AAP HU integration MUST support 480p at 30 FPS as the minimum default	For details, see Integrating Video.

¹ The USB-IF Battery Charging Specification v1.2 section 4.1.4 Shutdown Operation lists three methods of shut down. HU SHOULD avoid VBUS shut down as this also terminates the Android Auto connection.

<i>Audio</i>	• PCM (raw) audio RECOMMENDED; AAC-LC if necessary • Mixer to combine & route audio streams	PCM raw audio
<i>Human Machine Interface (HMI)</i>	Screen • 3.13" min height • 5.20" min width, assuming min height • 16-bit color (24-bit color RECOMMENDED) • Pixel Aspect Ratio between 0.9 - 1.1 Audio • Speakers (any) • Microphone Input • Dedicated "Voice" button (preferred on steering wheel) • "Answer Call" and "End Call" button (RECOMMENDED) • Master volume (physical) control • One (1) of the following input devices: ○ Touchscreen ○ 5-way (Dpad) ○ Rotary controller ○ Touchpad	Larger displays might require up-scaling of source video; see Matching Physical Display Size. Consult your Google technical contact early to confirm support for your display size. Support for controller-based inputs (such as rotary dials). Input hard-buttons map to corresponding Android Keycodes for media control or DPad. Additional (OPTIONAL) physical controls supported.

Screen Support

The screen height and width requirements in inches are valid based on current Android Auto projected UI constraints. While these are easy values to understand and measure and are useful for making quick decisions about compatible hardware, they are not definitive.

The definitive set of four criteria for screens that satisfy system requirements are as follows:

- Minimum horizontal width MUST be 748dp².

² Density-independent pixels (dp) defined at developer.android.com.

- Minimum vertical height MUST be 450dp.
- Minimum text size for secondary text on the Android Auto UI MUST be 12 arcmin. Ex: a capital H in the second line of text in a list is rendered at the secondary text size.
- 64dp tap target size MUST be ≥ 11.3 mm square. Applies to touchscreens only.

Partners MUST use the [Screen Size Calculator](#) in the Partner Toolkit to definitively determine compatibility. Input parameters are (a) diagonal screen size, (b) resolution, (c) pixel aspect ratio, and (d) viewing distance; output is the advertised dpi. For details on the importance of advertised dpi, see [Selecting Video Resolution](#).

The calculator contains JavaScript for demonstrating the calculations and algorithm used to determine Android Auto screen size compatibility.

Android Auto Projection Integration

To act as an endpoint, a head unit (HU) MUST include the AAP Receiver Library and implement the operating system (OS) Adaptation Layer that associates Receiver Library APIs to the native HU OS.

Receiver Library and Protocol Versions

This document covers version 1.2 of the AAP Receiver Library and AAP Protocol. Google recommends that partners integrate the most recent version of the Receiver Library available. Version 1.2 includes the following new features:

- Suppress status icons
- Report pixel aspect ratio and viewing distance
- Report any dead reckoning performed on GPS data
- Radio interface added

AAP Receiver Library

The AAP Receiver Library provides a full protocol endpoint solution in the form of a portable C++ library for the HU. The library handles the low-level protocol packet-framing, channel management, authentication, etc. However, it does not include OS-specific bindings required to complete the integration with relevant (video, audio, input, etc.) HU subsystems.

NOTE

The Receiver Library implements and encapsulates AAP interactions between the HU and mobile device (MD). To ensure compatibility with MD implementations, do not modify or re-implement this protocol.

In the source code distribution, the library is compiled as part of the Android build. Google also includes a standalone build file to compile the Receiver Library independently from the Android build system. Integrators MUST update the makefile to compile the Receiver Library with the same build tools and system libraries as required for the HU software build.

Open Source code is used by the AAP receiver in the following components:

- Protocol Buffers: code.google.com/p/protobuf/ (New BSD license)
- OpenSSL (not included in repo, but linked): www.openssl.org (BSD-like license)

OS Adaptation Layer

The OS Adaptation Layer connects the AAP receiver and the host OS for end-to-end integration. Integrators MUST implement the AAP receiver interface bindings and connect

each interface to the relevant HU subsystem; integrators might also need to recompile the Receiver Library with their build toolchain for compatibility with the HU OS environment.

Google provides reference implementations of the OS Adaptation Layer for Android, Linux, and QNX-based systems. Integrators are responsible for the commercial implementation of the OS adaptation layer, but may use these reference implementations as a starting point.

Software Requirements

The OS Adaptation Layer requires the following platform support.

- **C++ STL Support.** The base receiver library uses the following STL classes:
 - `Std::deque`
 - `Std::set`
 - `Std::string`
 - `std::vector`
- **Atomic Refcounting.** The receiver library uses atomic reference counting for memory management and requires access to an 'atomic increment and return' and 'atomic decrement and return'. The supplied implementation utilizes the gcc builtin `__sync_add_and_fetch()`.

AAP Receiver Interfaces

The majority of AAP integration work involves implementing receiver interfaces. The Integration Details section of this document provides an overview of the integration architecture, message flows, and interface definitions for the following receiver interfaces:

<i>Audio</i>	<i>Bluetooth</i>	<i>Input</i>	<i>Radio</i>	<i>VendorExtension</i>
<i>Authentication</i>	<i>HVAC (future)</i>	<i>Microphone</i>	<i>Vehicle Data</i>	<i>Video</i>

This document does not provide details for interface API definitions, method signatures and parameters. For these details, refer to the [AAP receiver library documentation](#).

Version Compatibility

Future versions of the AAP protocol may introduce new features supported only on MDs and HUs that implement the new AAP version. However, future AAP versions will maintain backwards compatibility with older versions wherever possible. If maintaining protocol backwards compatibility is not possible and an MD is incapable of running an earlier AAP version, the device will notify the user of the compatibility issue upon initial connection to the HU and immediately terminate the AAP connection.

Establishing AAP Connections

AAP connection establishment is the sequence of operations executed on the head unit (HU) and mobile device (MD) after the connection is initiated, typically when the user plugs the device into the HU.

Overview

When the user initiates connection, the MD executes a series of steps to prepare the connection with the vehicle HU. If a step is not necessary, the device skips that step (e.g. if the required applications are already installed and set up, the Manage Applications screen does not appear). Similarly, if a user has previously authorized a step, the device executes that step silently without user interaction.

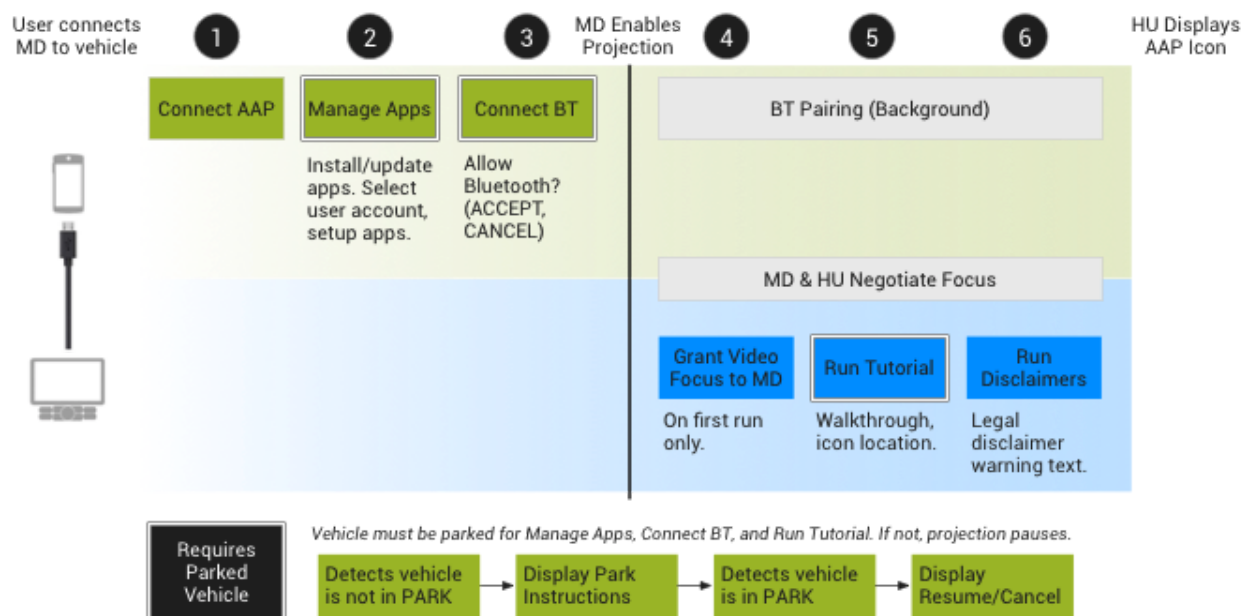


Figure 2. Overview of initial AAP connection establishment (known as *First Run*).

The initial AAP connection between a MD and HU is known as the *First Run* or *First Run Experience* (FRX). An AAP connection begins immediately when a user performs one of the following actions:

- Plugs the MD into the vehicle USB connection using a standard USB connector.
- Powers on the HU with the MD already plugged in. To avoid confusion, if the HU has a native setting option that allows users to enable/disable Android Auto connections, this setting **MUST** be enabled by default.

The HU initializes the USB host as an Android Open Accessory (AOA) device. For details on USB/AOA setup, see [Setting Up Transport](#).

The HU MUST be able to determine if the MD is compatible with AAP. This is done by initiating a AOAP connection and then listening for the correct signals:

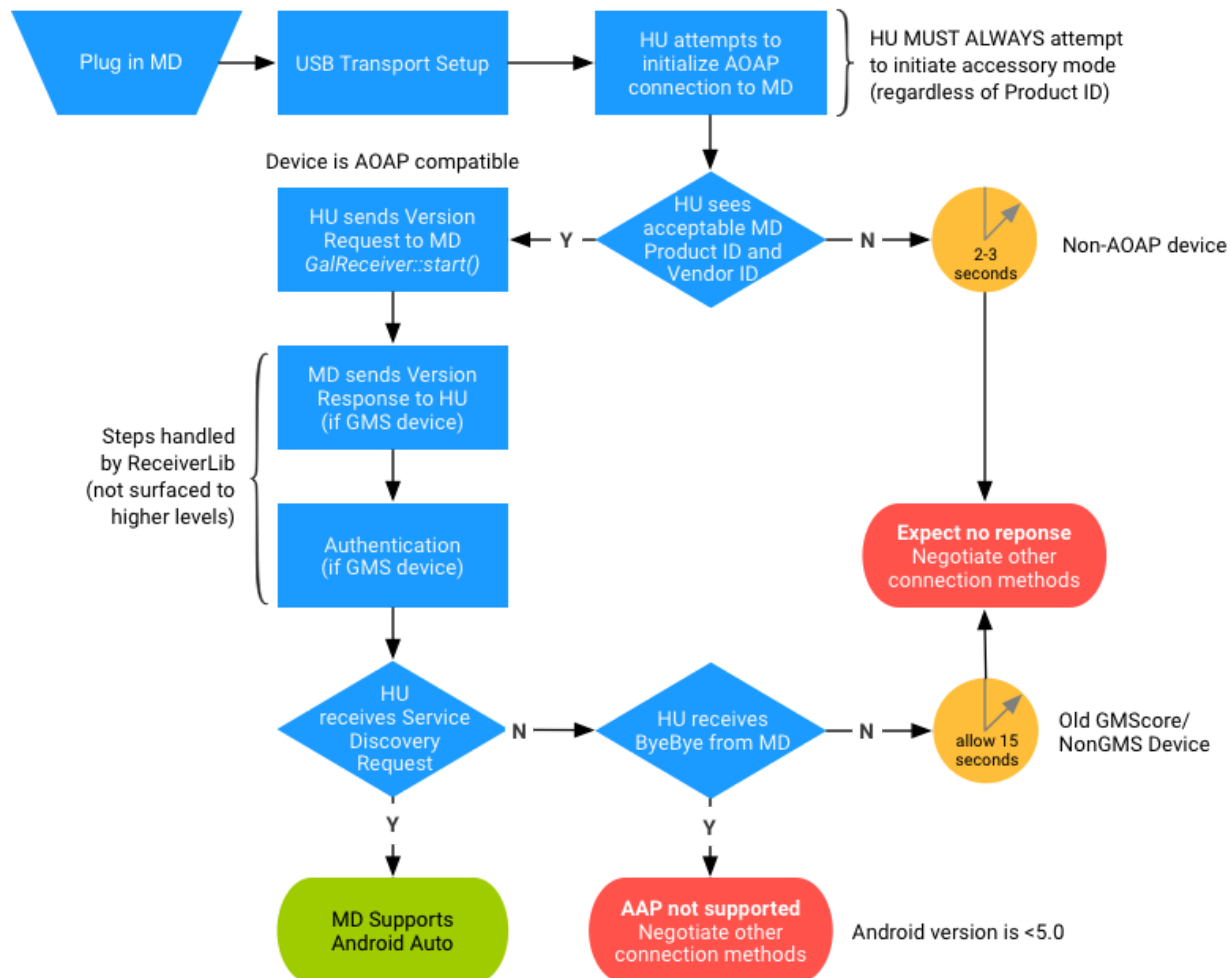


Figure 3. Overview of AAP connection flow.

AAP connections SHOULD persist as long as the MD is connected to the HU, or the MD terminates the connection. When the HU switches between native and projection mode, the AAP connection is kept active even if the AAP video or audio may not be rendered by the HU.

As the MD cannot know the state of other devices in the system, the HU MUST determine how to handle connections from multiple AAP compatible devices. For example, if there is a current active AAP connection between a MD and the HU, the HU implementation MUST account for a second AAP compatible device being connected. The first MD SHOULD continue to have control and additional devices SHOULD simply charge.

1 Connect AAP

The MD and HU authenticate using an [SSL certificate](#) and exchange AAP service discovery information.

After establishing an AOAP connection with the MD, the HU initiates negotiation of the AAP connection by sending a Version Request to the MD with ReceiverLib function `GalReceiver::start()`. The HU MUST NOT send additional Version Requests or ping requests (`GalReceiver::sendPingRequest()`) while awaiting a response from the MD. Encrypted packets MUST NOT be sent from HU to MD until authentication is complete.

If the MD supports AAP and authentication succeeds, the HU receives a Service Discovery Request, ie. `ReceiverLib IControllerCallbacks::serviceDiscoveryRequestCallback` is called. If the MD does not support AAP, then a ByeBye message MAY be received by the HU. Some MDs that do not support AAP will not send a Service Discovery Request *or* a ByeBye message; in this case the HU SHOULD reset the USB connection no less than 15 seconds after calling `GalReceiver::start()`.

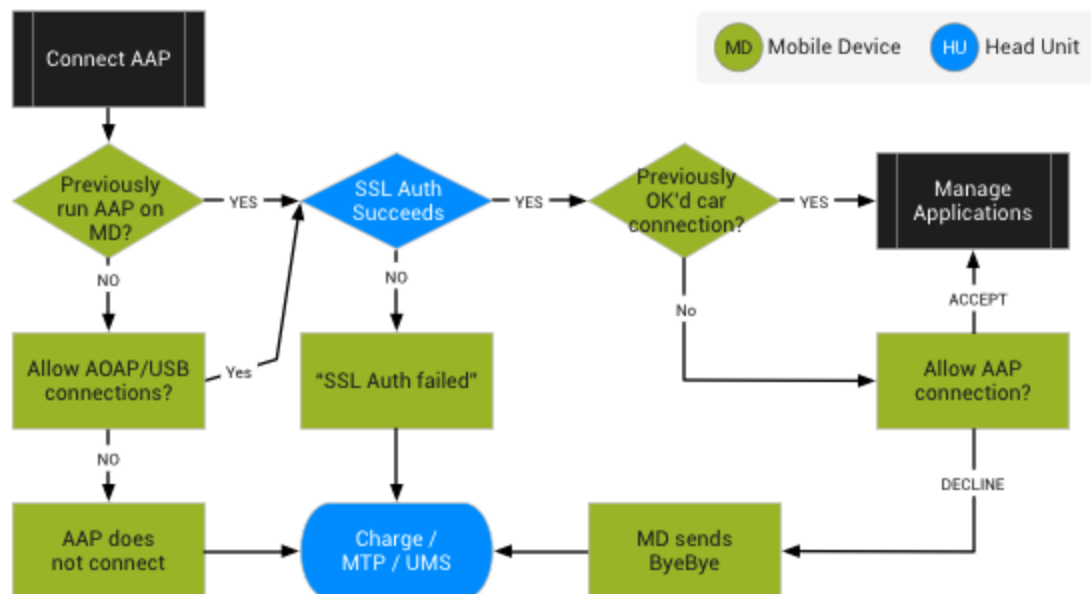


Figure 4. Authorizing AAP connection.

The Allow AAP Connection dialog prompts users to set preferences (OK, CANCEL) for the current connection and future connections (see Service Discovery for details on the Vehicle ID that triggers First Run Experience).

Service Discovery

During the authorization process, the HU sends metadata (relevant properties below) to the MD during service discovery. If properties are not basic strings (e.g. "2016" or "Galaxie

Sport”) and require decoding, the OEM MUST provide decoding information to Google.

Property	Required?	Description
<i>Make</i>	Required	Vehicle manufacturer brand.
<i>Model</i>	Required	Vehicle model (technical name suffices, but partner MUST provide Google with logic to parse name provided).
<i>Year</i>	Required	Model year, as distinct from year of manufacture (typically launch year).
<i>Vehicle_ID</i>	Required	Identifier for the vehicle, ideally regenerated for every factory reset performed on the HU. An MD connecting to a HU can determine whether this is the first time the MD has been connected to <i>this</i> HU by comparing vehicle_id. Do not use a traceable ID with real-world significance (such as the VIN). A valid <i>Vehicle_ID</i> SHOULD be unique string per HU, representing longer than 64 bits. For details, see Appendix A: Vehicle ID .
<i>Driver_Position</i>	Optional	Indicates the position (left, right, center) of the steering wheel, which may influence the position of UI elements.
<i>Head_Unit_Make</i>	Optional	HU manufacturer (supplier) name or brand.
<i>Head_Unit_Model</i>	Optional	HU model, identifier, or platform code.
<i>Head_Unit_Software_Build</i>	Optional	HU software build, typically an internal build identifier for HU software. If provided, it may be the same as the Head_Unit_Software_Version (below) or a different value entirely. Google issues a new SSL certificate for each compatible Android Auto implementation, so having this available in production is useful information but not required.
<i>Head_Unit_Software_Version</i>	Required	HU software version. Typically a version identifier a vehicle owner can find in the Settings menu of the HU.
<i>Session Configuration</i>	Optional	Values set to configure the MD behavior during

		projection (hiding clock, phone signal level and battery level indicators). HU can specify if the vehicle can play native media while running ASR. False when unspecified.
--	--	--

Uses and requirements for HU service discovery metadata:

- **OEM App Filtering.** These metadata are used by the Play Store to ensure that the correct OEM-authored apps are made available for installation. The values provided by the HU for some or all of these Service Discovery fields MUST therefore be *unique enough* to ensure the correct apps can be installed on the correct HU.
- **Targeted Disablement.** In cases where Google MUST disable AAP on a subset of AAP capable devices, the metadata fields are used to determine targeted HUs. The Make field must, at a minimum, be a stable and predictable value. For example, use a consistently cased OEM marque such as 'Car Company' rather than 'car_company' or 'CarCompany' or other variants. Additional fields may be used, but Make MUST be included and SHALL be consistent across Android Auto compatible products.

On first-run, the MD does not send video data until the user grants permission via the MD screen. Integrators may choose to draw the user's attention to MD screen with a message on the HU. A suggested implementation is:

1. After establishment of an encrypted AAP connection, start a timer for 3 seconds.
2. If the timer completes and `VideoSink::setupCallback()` has not yet been called (i.e. video feed not established), display a message instructing user to check the device.
3. When the MD starts to send video frames, the projected display MUST be shown without further user interaction.

Suggested text for the message is: "Android Auto can't connect right now. When it's safe to do so, check your Android phone."

② Manage Applications

The MD prompts the user to install, update, and configure required applications. This step occurs entirely on the MD and does not involve the HU. If all required applications meet minimum version requirements and are properly configured, this step is skipped.

③ Connect Bluetooth

When an MD is connected to a specific vehicle for the first time, the MD prompts the user to "always enable and pair with this car". If the user accepts, the MD silently connects to the vehicle whenever the MD is connected to the vehicle through AAP.

If the user gives consent, projection is enabled and the MD and HU establish a Bluetooth

connection that runs as a background task (for details on automatic AAP Bluetooth connections, see [Bluetooth](#)). If user cancels connection, AAP shuts down; on subsequent runs the First Run experience executes again.

4 Grant Video Focus to MD on request

The MD enables screen projection to the HU, enabling the HU screen to show projection contents when the MD requests video focus. Immediately after projection is enabled:

- For **First Run**, the MD requests video focus and the HU SHOULD grant it (as the end user is explicitly setting up Android Auto at this time).
- For **subsequent runs**, the MD and HU negotiate focus as detailed in [Audio Focus](#) and [Video Focus](#).

5 Run Tutorial

For first run the MD projects a tutorial that includes visible content and user interaction on the MD screen and on the HU screen.

6 Run Disclaimers

For first run, the MD projects disclaimers that require user consent.

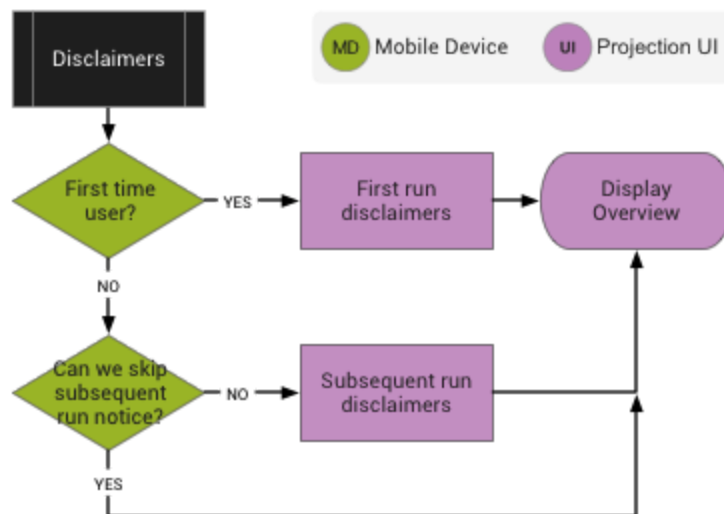


Figure 5. Disclaimers presented during first run user experience.

Without user interaction, the disclaimer MAY automatically timeout after displaying for a sufficient time.

Start Android Auto UI

The MD projects the Android Auto UI Home screen to the HU display.

Setting Up Transport

The AAP protocol requires a transport connecting the MD and HU. Current implementations support USB (and future versions may support Wi-Fi Direct) but the protocol is transport agnostic and can support any transport mechanism that provides sufficient bandwidth.

Universal Serial Bus (USB)

HU receiver implementations **MUST** support USB as a transport for AAP. In this configuration, the AAP protocol runs on top of the [Android Open Accessory Protocol](#) (AOAP) to communicate over USB bulk endpoints established between the MD and HU.

HU implementations **MUST** support **USB 2.0** and **AOAP 1.0 or 2.0**. The HU operates as a USB Host and AOA accessory. The HU powers the MD, which operates as a USB accessory.

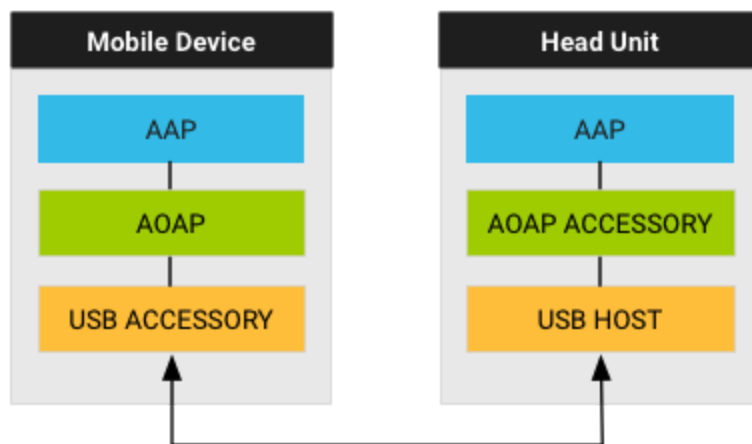


Figure 6. USB roles for mobile device and head unit.

Configuring AOAP

An Android MD identifies a vehicle as a AAP receiver based on accessory identifiers:

Manufacturer Name	Android
Model Name	Android Auto
Description	Android Auto
AAP Protocol Version	1.0
URI	<Empty>
Serial Number	<Empty>

Newer mobile devices might advertise as AOAP 2.0, which is backward-compatible with AOAP 1.0. For details on configuring an AOAP host accessory, refer to [Android Open Accessory](#).

USB Mode Selection

MDs and HUs can support multiple USB modes including charge-only, USB Mass Storage, MTP, and AOAP. Detecting AOAP/AAP based on connected device type is problematic; to prevent compatibility issues, the HU MUST attempt AOAP initiation regardless of connected MD USB Vendor ID (VID) and Product ID (PID). To correctly detect and negotiate USB mode selection, the host (HU) and client (MD) use the following process:

1. HU attempts to connect to MD using AOAP and advertise itself as an AOAP accessory .
2. If MD:
 - Supports AOAP, it successfully re-enumerates in AOAP mode.
 - Does not support AOAP, it ignores the AOAP handshake, enabling the HU to negotiate other USB modes or charge-only mode.

Head Unit Responsibilities

The HU implements the standard AOAP and discovery mechanism. On detecting a USB physical connection, the HU MUST switch the device to AOAP accessory mode (irrespective of initial VID/PID).

After the device is in accessory mode, the vendor ID is 0x18D1 (Google) and the product ID is 0x2D00 or 0x2D01.

NOTE

AOAP 2.0 adds extra product IDs for an OPTIONAL USB audio interface. If audio support is also enabled by the head unit, then product IDs 0x2D04 and 0x2D05 should also be supported. Android Auto requires only the *accessory* functionality.

If the MD supports AOAP, the HU establishes communication with the MD in USB accessory mode. For details on implementing the protocol and discovery mechanism, refer to [AOAP](#).

On initiating an Android Auto session, the HU MUST listen for an MD *disconnect message* (ByeBye) that indicates the user has exited or disabled AAP. On receiving the disconnect, the HU re-enumerates USB and does not send the request to start Android accessory mode. If the HU has not received an explicit user disconnect message for the session, it repeats the AOAP process for subsequent connections.

The HU can terminate the AOAP session with an MD by resetting the USB connection (and e.g. re-enumerate using MTP). USB reset is not intended for suspending the AAP communication; during normal operation, the USB AOAP session is maintained even if the

HU is not actively displaying projection content. For recoverable USB errors that cause a temporary interruption in projection, AAP automatically takes video focus on connection stabilization.

Mobile Device Responsibilities

The mobile device is responsible for:

- Switching to AOAP mode when requested by the HU.
- Responding to HU initiation of Android Auto connection with an appropriate handshake.
- Registering and responding to HU Service Discovery declarations during Android Auto connection setup.

After an AOAP connection is established, the HU uses the Android Auto Receiver Library to establish a secure, encrypted communication channel (see [Authenticating](#)).

USB Charging

HUs MUST support charging-only mode. When an HU USB port is operating as a standard downstream port and an MD selects charging-only mode, the HU sends a USB SET_CONFIGURATION request that allows the MD to draw current up to 500mA.

The HU MUST also support CDP as defined in the [USB-IF Battery Charging Specification v1.2](#).

Authenticating

The AAP protocol uses industry standard [TLS 1.2](#) with Client Authentication to secure communications between the head unit (TLS Client) and mobile device (TLS Server). The first step in establishing a AAP connection is mutual authentication between the MD and HU following the [TLS Client-authenticated TLS handshake](#) protocol.

Authentication Certificates

The AAP sender (on the MD) and the AAP Receiver Library (on the HU) perform the core mutual authentication protocol. If authentication succeeds, the sender and receiver library initiate the AAP projection connection. If the receiver library cannot verify the sender certificate, the HU terminates the connection and disconnects AOAP (the MD might attempt to reconnect in another USB mode).

The receiver library on the HU performs the entire certificate verification. While integrators do not need to implement this authentication, they MUST ensure the HU securely stores keys and passes those keys to the receiver library on request. Software-backed security (such as Linux file system permission control) is sufficient, but hardware-backed secure storage is strongly RECOMMENDED.

The HU and MD establish mutual trust using two (2) certificates signed by the Google CA:

- HU certificate sent to MD, which verifies it to ensure the HU is a valid AAP receiver.
- MD certificate sent to HU, which verifies it to ensure the MD is a valid AAP device.

Google provides HU certificates to integrators but recommends that integrators request and use a unique client certificate for each make/model/year (data not encoded in certificate).

Public/Private Key Format	RSA with 2048 bit keys
Digital Certificate Format	X.509

Authentication Sequence

The AAP Receiver Library uses the OS Abstraction layer API to retrieve the HU digital certificate and private key. After authentication completes, the receiver library notifies the HU of the MD identity and initiates the AAP protocol.

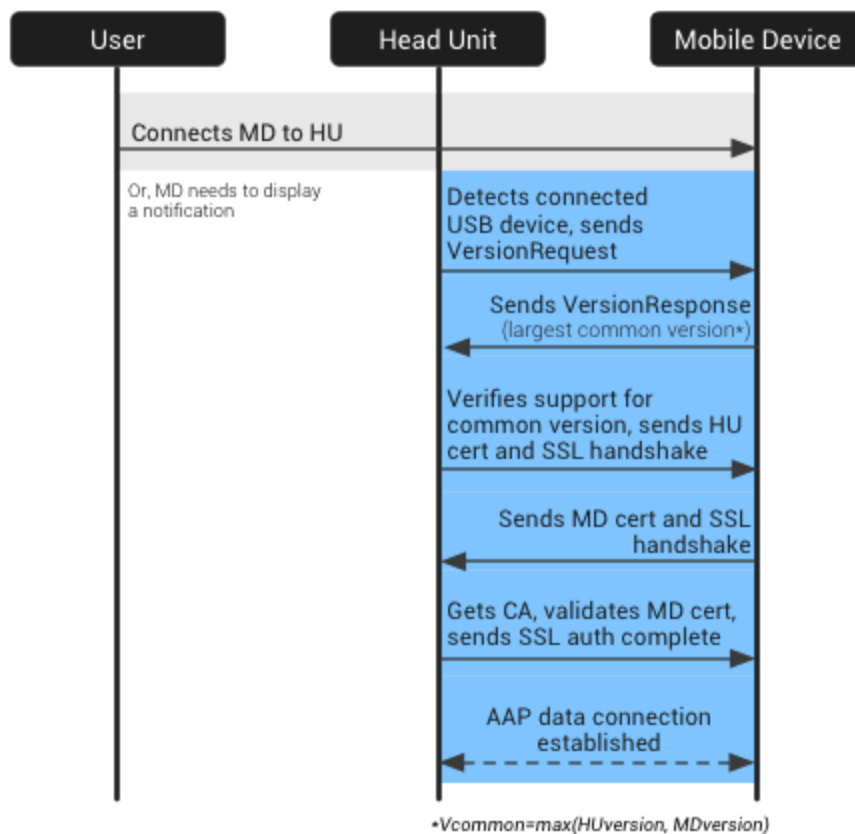


Figure 7. Mobile device and head unit establish trust using SSL certificates.

SSL authentication might fail with error “Certificate not yet valid” or “Certificate expired” if the internal clock on the HU is incorrect and/or does not match the approximate time on the MD. To resolve, the HU MUST obtain the correct time (GPS, Mobile Network/NITZ, NTP, etc.).

Setting Up Bluetooth

To provide a consistent hands-free telephony experience across both the native and projection user interfaces, AAP uses Bluetooth Hands-Free Profile (HFP) for voice telephony communication. However, AAP does not use Bluetooth for speech recognition (BVRA) or media playback audio streaming (A2DP).

While AAP needs a Bluetooth HFP for telephony to function during an active AAP session, the absence of an active HFP connection will not prevent the AAP pairing process from completing. The HU MUST support Bluetooth HFP, but not every MD will always have an active Bluetooth connection. For example, the user may have explicitly turned off Bluetooth. AAP does not disable other protocols such as RFCOMM.

Bluetooth Pairing & Connection

After establishing AAP connection, the MD can automatically pair with the HU over Bluetooth.

If the MD has **never paired** with the HU before, the HU and MD use the following Bluetooth pairing and connection flow:

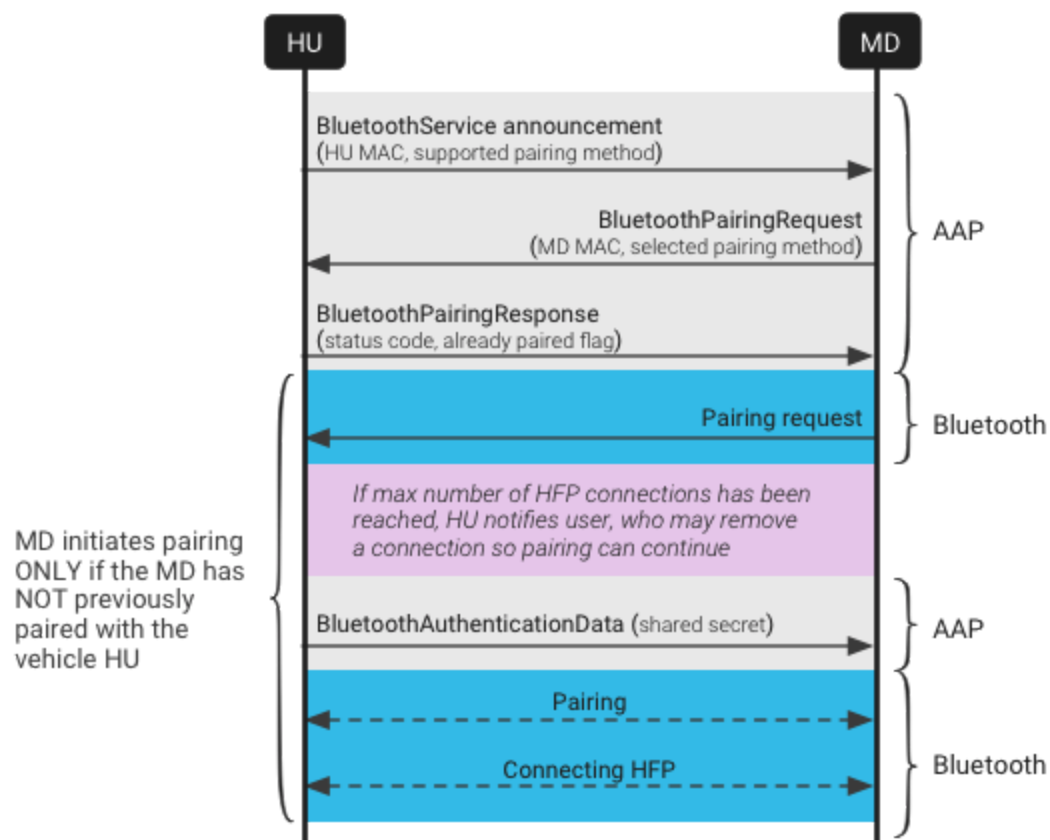


Figure 8. Bluetooth pairing authentication and connection flow when MD has never paired with vehicle.

If the MD has **previously paired** with the HU, the HU and MD use the following Bluetooth pairing and connection flow:

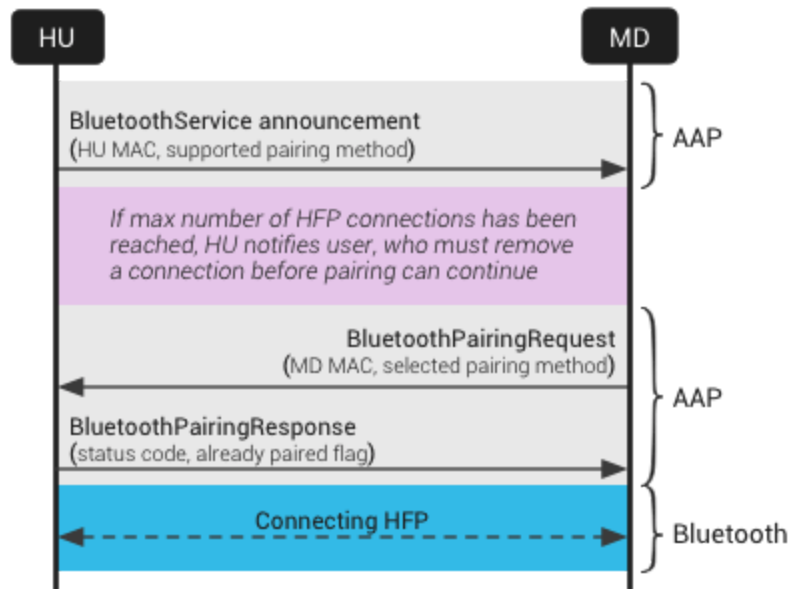


Figure 9. Bluetooth connection flow when MD has previously paired with vehicle.

BluetoothService Announcement

During the service discovery phase, the MD looks for a BluetoothService announcement from the HU, which SHOULD include the:

- **HU Bluetooth MAC address.** If the HU wants the MD to skip the Bluetooth pairing and connection process, the HU can declare its Bluetooth MAC address as SKIP_THIS_BLUETOOTH (a useful trick when development and debugging).
- **HU supported Bluetooth pairing methods.** AAP supports both PIN and numeric comparison as pairing methods.

Protocol Messages for Bluetooth Pairing

After the HU provides a valid Bluetooth MAC address and supported pairing methods, the MD and HU use the following AAP messages to establish Bluetooth pairing and connection.

Message	Description
<i>BluetoothPairingRequest</i>	Sent by MD to request exchange of Bluetooth pairing information; can resend this message anytime to fix a corrupted state. Receiving HU MUST send BluetoothPairingResponse within 1 second. This message includes the device Bluetooth MAC address and

	the pairing method that will be used by the device.
<i>BluetoothPairingResponse</i>	Sent by an HU in response to BluetoothPairingRequest. Because requests may be re-sent anytime by an MD attempting to fix a corrupted state, a HU MUST be able to handle multiple BluetoothPairingRequest messages in the same AAP connection. An HU MUST send a response within one (1) second of receiving the BluetoothPairingRequest. If the HU can be ready to pair (using the pairing method specified in the request) within one (1) second of receiving the BluetoothPairingRequest, it sends a BluetoothPairingResponse with STATUS_SUCCESS as the status. • Readiness. To be ready, the HU MUST set the AAP MD priority as the highest and make the HU Bluetooth module discoverable by the MD. The HU MUST NOT make any HFP connections with other (non AAP) devices until the AAP connection ends. • Previously Paired. An HU MUST send a BluetoothPairingResponse even when already paired with the MD (necessary when the MD loses pairing information and must re-initiate pairing). • Success. When the HU sends a BluetoothPairingResponse with STATUS_SUCCESS as the status code, the HU MUST include <i>already_paired</i> field in the message. This field value MUST be true when the HU already has a link key (pairing information) for the Android Auto connected device (vehicle believes it is already paired with the device). • Pairing. After receiving a BluetoothPairingResponse with STATUS_SUCCESS as the status code, the MD initiates the pairing and connection process with the HU. If each side has the link key (pairing information) for the other side, the pairing step is skipped and the MD initiates the HFP connection only. If the HU cannot be ready to pair within one (1) second of receiving a BluetoothPairingRequest, it SHOULD send a BluetoothPairingResponse with STATUS_BLUETOOTH_PAIRING_DELAYED as the status. When the MD receives this message, it ignores the <i>already_paired</i> field and assumes the HU will send another

	BluetoothPairingResponse with STATUS_SUCCESS as the status.
<i>BluetoothAuthenticationData</i>	Sent by an HU to provide authentication data for Bluetooth pairing (after MD initiates pairing). The MD uses the data to complete pairing. If data is valid, the MD and HU pair and connect to HFP. Auth data is a UTF-8 character encoded string containing the passkey or numeric comparison value.

After pairing, the HU can request the Bluetooth Phone Book Access Protocol (PBAP). If the user opts into PBAP, the HU Bluetooth module MAY synchronize the local address book data with the MD contact database. However, the HU MUST coordinate the AAP and PBAP initialization and present a sensible UI (i.e. no overlapping windows for pairing permissions).

The HU MUST handle the following edge cases during pairing:

- **HU connected to a separate MD engaged in a Bluetooth call.** HU MUST return STATUS_BLUETOOTH_PAIRING_DELAYED in a PairingResponseMessage to indicate the requesting MD SHOULD wait for a response. After the call ends, the HU sends a BluetoothPairingResponse message with STATUS_SUCCESS to the requesting MD to indicate the HU is ready to accept a call and in Bluetooth discovery mode.
- **Upper limit of connected Bluetooth devices reached blocks AAP pairing.** Many HUs have an upper limit to the number of supported paired BT devices. The HU might block an otherwise successful AAP connection because the paired device limit has been reached. To avoid blocking AAP connections based on device limit:
 - We STRONGLY RECOMMEND that an HFP connection is established.
 - We STRONGLY RECOMMEND automatically dropping the least recently used pairing.
 - The HU MAY display a message to users informing them a connection was removed to allow pairing to complete.
 - The HU MAY walk the user through dropping at least one already-paired device so AAP connection can continue.
 - If an AAP connection was prevented due to reaching the paired device limit, the HU MUST notify the user of the maximum number of paired devices are already used on the HU.
- **Non-AAP MD attempts to pair.** HU MUST allow pairing with the AAP connected MD but MAY choose to allow or disallow pairing with other MDs.
- **Bluetooth disabled on HU.** HU MUST enable Bluetooth and pair requesting MD.

HFP Connection

After pairing, the MD attempts to connect to Bluetooth Hands-Free Profile (HFP). If the HU can support only one HFP connection at a time, the HU MUST allow connections to the AAP connected MD:

- If Bluetooth connection process is in progress before projection mode begins, the HU MUST continue the connection process after switch to projection mode.
- Projection mode telephony functions MUST be disabled on the MD and HU until the HFP connection is established.
- Regardless of whether HU is connected to another device, the HU MUST allow a Bluetooth connection with the MD as soon as possible (see edge cases below).
- MD cannot connect to other device for HFP while AAP is connected.
- HU SHOULD suppress UI elements exposing PBAP or MAP when AAP is connected, and cannot connect to other device for HFP while AAP is connected.

The mobile device MUST handle the following use cases when connecting to Bluetooth:

- **MD playing audio.** If the MD is connected to an A2DP device (including vehicle), AAP leaves the A2DP connection intact. As with other HU media streams such as FM radio, AAP expects the HU to correctly manage audio focus with an active A2DP device. A2DP SHOULD be treated by the OEM as another source, such as a USB stick or radio. For example, if the user chooses to play music via Android Auto, and was previously streaming from an A2DP source, the HU SHOULD grant the request from Android Auto to change audio focus to Android Auto. For details, see [Audio Focus](#).
- **MD on a non-Bluetooth call.** MD connects to the HU and routes the call over Bluetooth.
- **MD on a Bluetooth call.** MD continues Bluetooth call and displays projection mode.
- **MD on a call to HFP device.** MD continues the call, disconnects from the other HFP device, and connects to HFP on the vehicle.

Additional Bluetooth Profiles

Android Auto uses only Bluetooth HFP. However, during an Android Auto session, the HU MAY maintain one or more of the following additional Bluetooth profile connections:

- Message Access Protocol (MAP)
- Phonebook Access Protocol (PBAP)
- Personal Area Network (PAN)
- Remote SIM Access Protocol (RSAP)

Such connections MUST be managed so they do not interrupt the Android Auto experience.

Bluetooth Reconnection

If Bluetooth disconnects or is lost during an active AAP connection, the MD continuously attempts to connect to HFP. The HU MAY (but is not required to) attempt to connect to HFP, but MAY NOT connect to another Bluetooth device; both MD and HU MUST disable all projection mode telephony functions until HFP reconnects. If Bluetooth pairing is lost or reconnect attempts repeatedly fail, the MD may re-initiate Bluetooth pairing after exchanging BluetoothPairingRequest/BluetoothPairingResponse messages.

Integrating Video

This section provides a technical overview of AAP video integration (for details, refer to the VideoSink in the [AAP receiver library documentation](#)). In projection mode, the mobile device (MD) controls video on the head unit (HU) screen and displays a video stream encoded by the MD. The HU MUST display projected video full-screen on the primary center console display.

Video Format	H.264 elementary byte stream
Profile	Baseline Profile

During initial service discovery, the MD and HU negotiate the screen resolution, density, pixel aspect ratio (PAR), and maximum frame rate of the primary display. The MD renders and encodes content in the negotiated configuration (no upscaling or downscaling required). We strongly recommend HUs use a hardware video decoder capable of rendering full-screen video of at least 60 frames per second (FPS) to match the typical mobile device 60 FPS UI redraw rate. For lower end hardware, HUs can specify a 30 FPS video stream frame rate. The higher the frame rate, the lower latency for input events (especially for touch gestures).

To preserve bandwidth and power, the AAP protocol sends video frames only for projection UI updates (and does not send frames when no updates are available). However, HU video decoders MUST avoid adversely affecting latency by not entering power-saving mode even when not receiving frames.

At runtime, MD selects from supported resolutions and frame rates to determine actual frame rate. Selection depends on many factors such as device power state and is the reason for the required resolution of 800x480 at 30 FPS. Frame rate may change either continuously or in discrete jumps to optimize system performance, e.g., from 30 FPS to 29 FPS, or directly from 15 FPS to 5 FPS.

NOTE

H.264 video sent from MD contains only I frames and P frames (buffering set to minimum). HU SHOULD decode each frame without waiting for additional frames.

Selecting Video Resolution

AAP MDs support the following codec resolutions; HU video decoders **MUST** support at least 800x480 and **OPTIONALLY** more of the following:

Codec Resolution	Max Video Bit Rate (kbits/s)	Max Frame Rate
800x480	4,000	60 FPS
1280x720	6,000	60 FPS
1920x1080	8,000	60 FPS

Because hardware video encoders/decoders are optimized to support a set of fixed video resolutions, using non-standard codec resolutions can result in significant performance loss.

The MD and HU negotiate video resolution during the initial AAP protocol handshake:

1. During AAP service discovery, the MD requests the VideoConfigurations supported by the HU.
2. After establishing the video channel, the HU sends a Config Message with a prioritized list of indices (most preferred first) for the previously-provided VideoConfigurations.
3. MD selects a configuration based on HU preference and device hardware encoder capabilities.
4. MD sends a Start message (with the index of the selected configuration) to HU.

Request the Highest Resolution Possible

The HU **SHOULD** request the highest resolution video that the protocol supports and gracefully handle all lower resolutions. For example:

- If the native screen is 1920x1080, the HU **SHOULD** advertize support for 1080p, 720p, and the required 480p.
- If the native screen is 1024x720, the HU **SHOULD** advertise support for 720p with appropriate pillarbox margins and the required 480p.

Not all MDs support encoding video at 1280x720 or 1920x1080 codec resolution. However, more MDs will support generating higher resolution video streams and HUs **SHOULD** anticipate supporting high resolutions.

Selecting a Density to Report

Each VideoConfiguration contains a density. Every screen has a physical density expressed as the number of physical pixels in each inch, known as dots per inch (DPI).

In Android, displays are bucketed into one of six generalized densities³; previously, Google encouraged Android Auto partners to report VideoConfiguration density as one of these densities, most commonly 160 or 240. It is now REQUIRED to report the preferred density calculated by the Screen Size Calculator (for details, see [Head Unit Requirements](#)).

NOTE

The reported density from the Screen Size Calculator optimizes the Android Auto UI (in density-independent pixels), font size for maximum legibility given a viewing distance, and tap target size. It does this while minimizing pillarboxing or letterboxing of the UI inside the video stream.

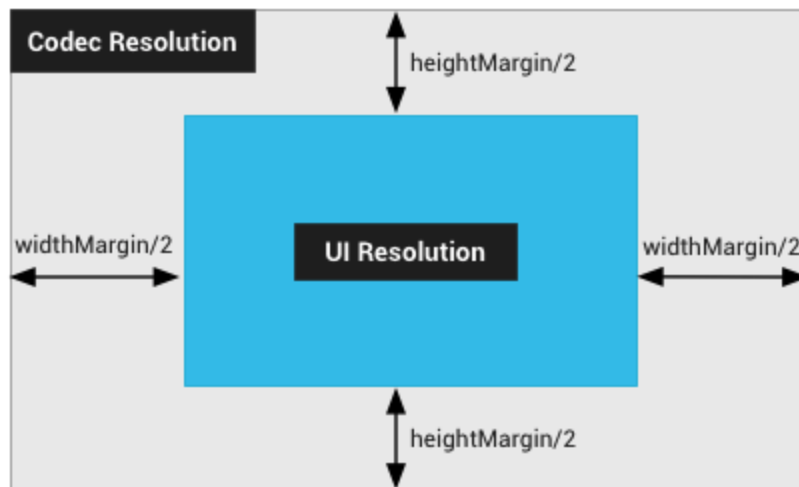
Matching Physical Display Size

HU displays rarely have standard physical sizes or match AAP supported resolutions or aspect ratios, making a 1:1 rendering of the projection UI to the HU display impractical. Displays that use non-square pixels SHOULD use AAP v1.2 (or higher) and report pixel ratio to the MD, which can apply correction on the picture sent to the HU to account for non-square pixels.

To support a wide range of in-vehicle display hardware, AAP uses a *logical UI resolution* that is always less or equal to the codec resolution of the transported video stream. MDs render content only in the area specified by the UI resolution, with horizontal and vertical margins padded (using pre-set buckets of specific codec resolutions) to match the logical UI resolution.

When the MD encodes a video stream, it fills horizontal and vertical margins with an arbitrary color. The OS abstraction layer decodes the stream, clips it to the UI resolution size, and discards the margins:

³ [Developer.android.com](https://developer.android.com) has details on screen density support in the Android platform.

**NOTE**

The MD may not completely fill the UI Resolution area with visible UI and instead place black bars at the top and bottom (letterbox) or at the sides (pillarbox).

Integrations **SHOULD NOT** attempt to account for such bars as they may change over time.

Figure 10. Codec-to-UI resolution.

Required Video Stream Support

Many MDs are limited to output 800x480 H.264 video at 30 FPS. All HU integrations are therefore **REQUIRED** to handle 800x480 video stream in addition to higher resolution configurations.

To display 800x480 projection on a higher resolution HU screen, the integrator **MAY** letterbox the content (i.e., display at original resolution with appropriate margins) or up-scale the projected feed to a higher display resolution while maintaining aspect ratio. Such integrations **MUST** be discussed with Google at earliest opportunity to confirm support.

Video Focus

AAP enables users to seamlessly switch back and forth between the native UI and projection UI, requiring a robust video focus negotiation model. However, because the main console display is also used for safety-critical features (such as a backup camera), the **HU always controls video focus** and determines when the MD can project content to the screen.

AAP includes two (2) main focus modes, *Projection* and *Native*. For transient native screens (such as turn-signal activated blind spot cameras expected to show for a short time only) a third focus mode is available: *Native Transient*. From the HU perspective, Native Transient is the same as Native; the purpose is to provide behavioral hints to the MD. In addition, transient Native content **MAY** overlay the Projection video.

During connection setup video focus MUST NOT be granted to the MD by the HU (VIDEO_FOCUS_PROJECTED) until `VideoSink::setupCallback()` has been called. Attempting to grant video focus to the MD before or during `VideoSink::setupCallback()` will cause a failure in the `VideoSink`.

Mode	MD sends video	HU sends touch/controller events
Projection	Y	Y
Native	N	N
Native Transient	N	N
Overlay	Y	For details, see Overlay

Projection Mode

In projection mode, the HU renders a fullscreen video stream from the MD and forwards all touch/controller events to the MD. This is the preferred mode for active AAP connections.

Native Mode

In native mode, the HU renders the full-screen native UI (i.e. backup camera) and the MD does not stream video to the HU. After native mode interactions complete, the HU SHOULD return video focus to projection. If projection is active but HU has video focus, the HU MUST send the MD appropriate key events (launch Google Search from native, KEYCODE_MEDIA/GUIDANCE, forwarded from native).

Overlay

HUs can display short, transient overlays on top of projected video content. If the overlay:

- Has simple straight-line boundaries and obscures less than 30% of the screen area, input events MAY be sent to Android Auto from the unobscured screen region only.
- Obscures large portions of the screen or has complex boundaries, input events MUST NOT be sent to Android Auto from anywhere on the screen.

Mode Examples

After establishing an AAP connection, including the MD announcing the video encoder is ready (listen for `IVideoSinkCallbacks::setupCallback()`), the HU remains in native mode but the MD can request video focus immediately. The HU MUST notify the MD immediately when video focus changes.

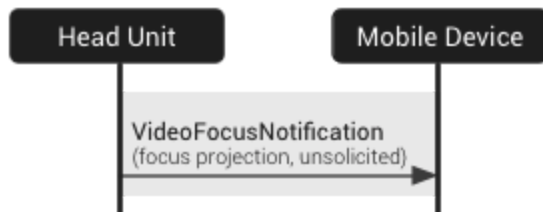


Figure 11. Head unit notifies mobile device of video focus grant.

HUs SHOULD use native mode only to perform functions not available in projection mode; if an HU can provide native functions using a video overlay then AAP maintains video focus. For example, if a user presses a climate control hard button to display overlaid video (e.g. updated temperature set point), the HU SHOULD NOT switch to native mode to display the climate control UI. If the climate control UI is a modal dialog overlaid on top of projection mode, the HU SHOULD direct user inputs to the native UI (and not the projected UI).

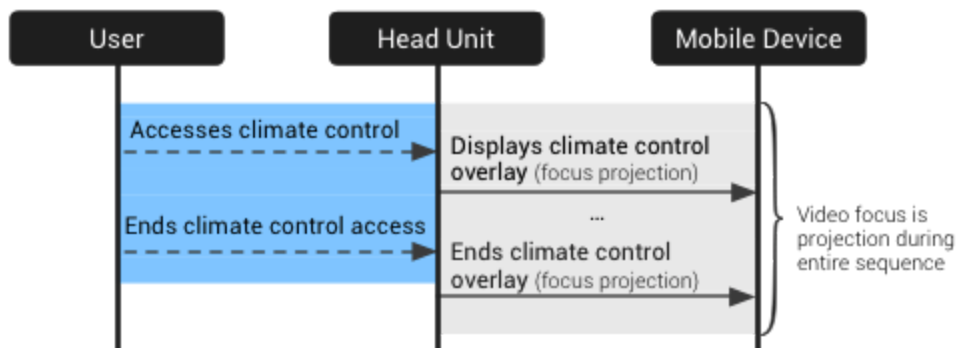


Figure 12. User accesses climate control.

Video Focus Requests

The MD can send video focus requests to the HU to request video focus be granted to projection mode. In general, the HU SHOULD grant video focus requests unless it is unsafe to do so (i.e. vehicle is in reverse and the HU MUST show backup camera). The MD sends video focus requests only in response to a direct user action, such as key input or voice command.

In response to a video focus request, the HU sends a video focus notification to the MD, indicating the new video focus mode.

For example, the following flow shows a user issuing a speech recognition request to the MD:

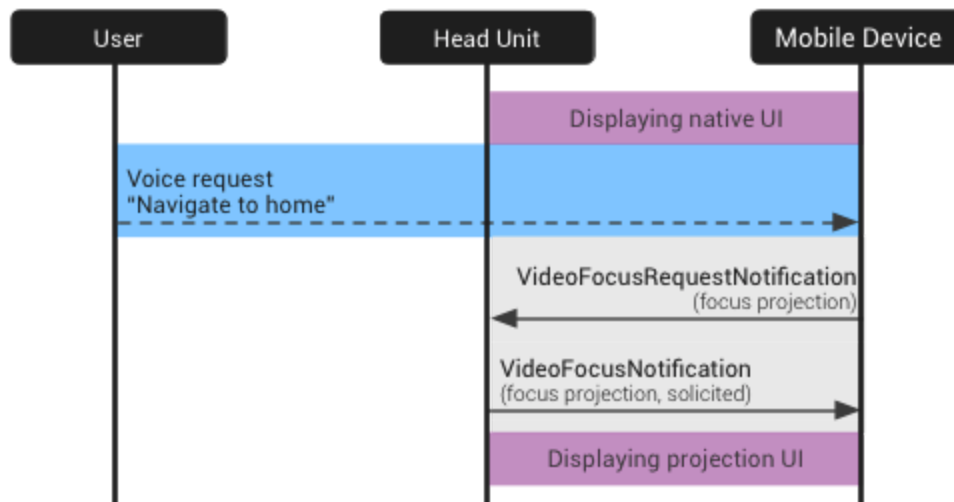


Figure 13. User requests MD to “navigate home”.

The HU might not immediately honor a request for video focus, such as when the vehicle is in REVERSE and the HU is displaying the backup camera. In such cases, the HU does not deny the focus request but delays granting video focus to projection mode until after the backup camera is no longer required (i.e. vehicle is in DRIVE).

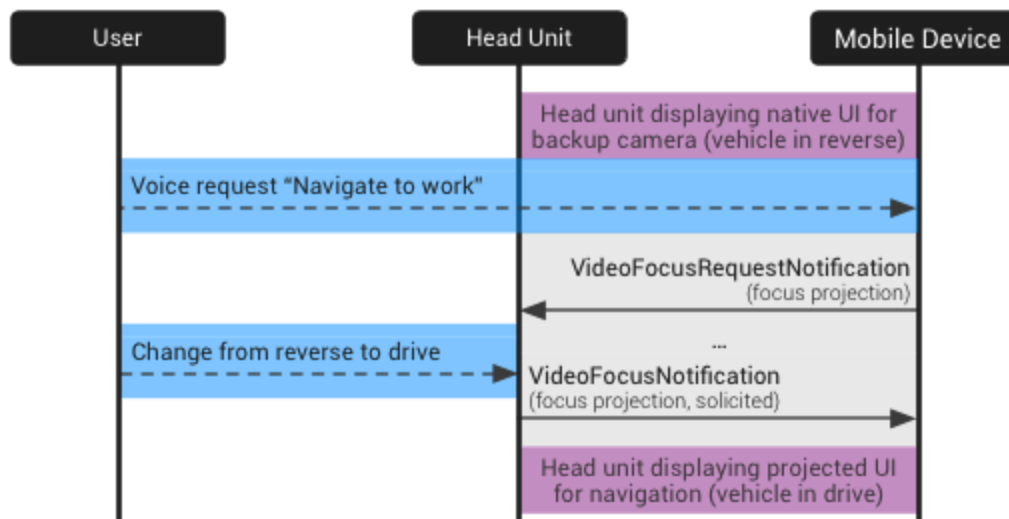


Figure 14. User requests MD to “navigate work” while native reversing camera in use.

Latency, Flow Control & Clock Skew

AAP video streams are real-time, enabling the HU to render streams immediately. However, streams use an n-ACK based flow control scheme to enable the video source to throttle throughput when under load. AAP considers two (2) latencies: video setup and video output:

Video setup latency (max)	500ms
Video output latency (max)	50ms

Video *setup latency* is the time required by the vehicle to establish the video stream and includes the time necessary to perform any on-screen transition to projection mode and power on the primary display.

Video *output latency* is the delay between a video frame being made available from the AAP receiver and reproduction of the video on the primary display. Output latency includes video buffering performed by the in-vehicle graphics pipeline, video codec decoder latency, and display latency.

The user-visible video latency is higher than the listed video output latency because the latter does not include encoder and transport delays.

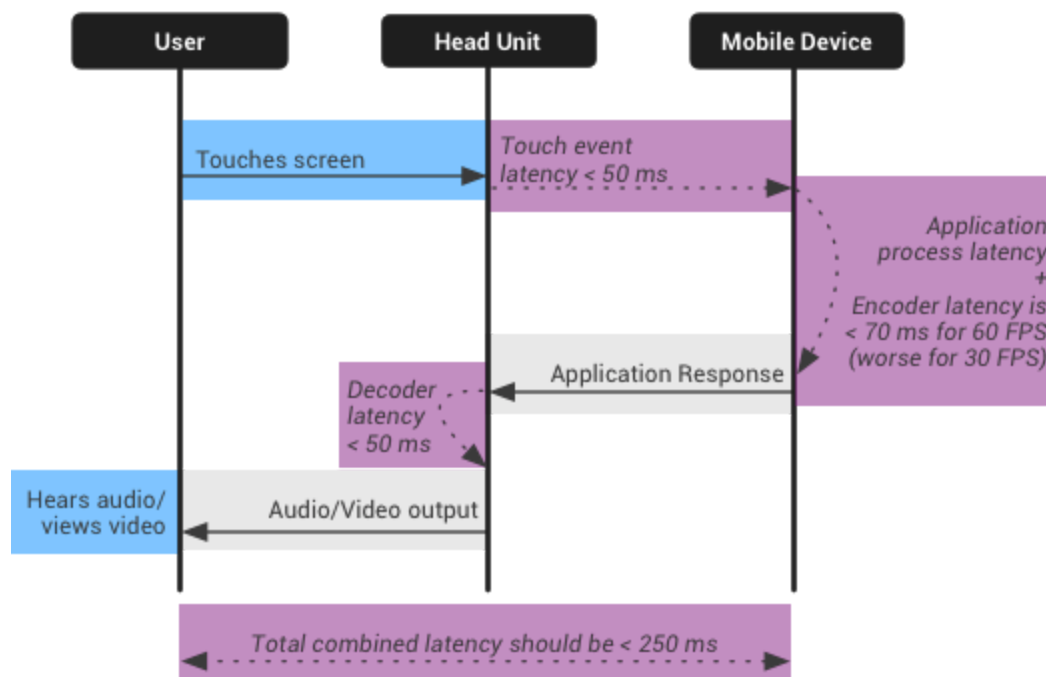


Figure 15. Video latency (listed + user-visible).

Because the MD does not render continuously, the HU MUST report the depth of its decoder pipeline so the MD can send enough idle frames to ensure the display updates properly. This means decoder depth increases MD power consumption in addition to increasing latency.

Integrating Audio

This section describes AAP audio architecture, audio streams, audio focus management, and audio management policy. Modern vehicles use deterministic audio policies to manage multiple audio streams. AAP introduces additional audio streams from mobile device (MD) applications, so integrators **MUST** ensure their existing architecture and audio policy successfully manages streams across the MD and the vehicle head unit (HU).

This section uses the following terminology:

- **Audio Source** (*output*). Logical endpoint that generates audio data and sends to an audio stream.
- **Audio Sink** (*input*). Logical endpoint that receives audio content from an audio stream (e.g. in-vehicle speakers).
- **Audio Stream**. Named combination of an audio source and audio sink.

Audio Streams

Output audio streams are sent from the MD to the HU; *input audio streams* are sent from the HU to the MD.

AAP includes one (1) audio input stream for **Microphone**, one (1) bi-directional audio stream for **Legacy In-Call** Bluetooth audio, and four (4) audio output streams:

Channel	Stream*	Purpose
UI	AUDIO_STREAM_SYSTEM	user interface feedback
Guidance	AUDIO_STREAM_GUIDANCE	driver guidance
Voice	AUDIO_STREAM_TELEPHONY	conversational voice
Media	AUDIO_STREAM_MEDIA	cabin media

* As defined in the receiver library *AudioSink.h*

Streams transmit audio to and from the vehicle using the following architecture:

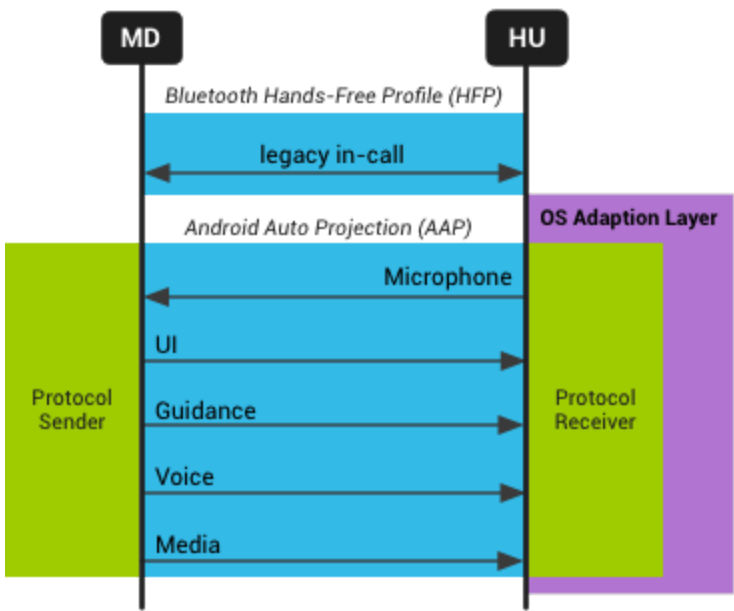


Figure 16. Audio stream architecture.

Streams use the following transports and codecs:

Stream	Direction	Transport	Protocol ⁴
UI	device-to-vehicle	AAP (future)	PCM (REQUIRED) and AAC-LC (OPTIONAL) 16 bit, 16 kHz, mono
Guidance	device-to-vehicle	AAP	PCM 16 bit, 16 kHz, mono
Voice	device-to-vehicle	AAP (future)	PCM (REQUIRED) and AAC-LC (OPTIONAL) 16 bit, 16 kHz, mono
Media	device-to-vehicle	AAP	PCM (REQUIRED) and AAC-LC (OPTIONAL) 16 bit, 48 kHz stereo
Microphone	vehicle-to-device	AAP	PCM 16 bit, minimum 16 kHz, mono
Legacy In-call	bi-directional	Bluetooth SCO	HFP 1.5 (SCO/CVSD)

UI (device-to-vehicle)

The UI feedback audio output stream is intended for short audio feedback (beeps, chimes, etc.) in response to UI input. This audio is typically mixed with other audio playback at constant volume without pause or ducking, and can be mixed on top of other streams. Additionally, this audio is often intended for the driver and can be routed to audio channels

⁴ AAC-lc format is: RAW AAC frame without header, decoded buffer size in one frame: 1024 samples for 16 kHz, 2048 samples for 48 kHz.

aimed at the driver. The HU decides which output channels receive this audio stream; the MD does not send focus requests for this stream. For details, see [Audio Focus Policy](#). AAP does not currently use the UI stream.

Guidance (device-to-vehicle)

The guidance audio output stream plays high-priority transient audio such as traffic warnings and spoken turn-by-turn navigation instructions. This audio is often intended for the driver and can be routed to audio channels aimed at the driver. The HU decides which output channels receive this audio stream.

Voice (device-to-vehicle)

(Not currently supported) The voice stream is designed to handle telephony and the voice audio output stream plays conversational voice audio such as voice call output. This audio is often intended for the driver and can be routed to specific audio output channels aimed at the driver. The HU decides which output channels receive this audio stream.

NOTE

Guidance and *Voice* are distinct streams because the vehicle HU SHOULD mix the Guidance stream at a higher priority.

While future versions of AAP may use the voice stream for telephony, current implementations use Bluetooth for telephony (see [Legacy In-Call](#) stream).

Media (device-to-vehicle)

The media audio output stream plays low-priority, long-form media from applications on the MD such as music, audiobooks, podcasts, Internet radio, etc. This audio is typically intended for the entire cabin, and SHOULD be routed to the main cabin audio sink. This audio source SHOULD never play on top of any other source and focus MUST be mutually exclusive with vehicle media. For details, see [Audio Focus Policy](#).

Microphone (vehicle-to-device)

The microphone audio input stream captures microphone audio using the built-in vehicle microphones and transmits that audio to the MD, which can use it for voice calls, speech recognition, etc.

Legacy In-Call (bi-directional)

In-call voice audio is currently routed over a Bluetooth HFP/SCO link instead of the AAP protocol. In future mobile device implementations, the voice call audio stream will route over the microphone (input) and voice (output) streams over the AAP connection. Negotiation of the voice call audio link is performed using Bluetooth Hands-Free Protocol (HFP) and is not detailed in this document.

Audio Focus

Audio focus determines which audio sources can play through which streams to the audio sink at a given point in time. If a source has focus, then its audio is routed to the audio sink.

- Audio focus **requests** inform the HU the MD wants audio focus. An MD sends a request when it wants to play audio.
- Audio focus **notifications** inform the MD of the current audio focus state. An HU sends a notification in response to audio focus requests or unsolicited (prompted by external events).

Android Auto integrators **MUST** consider changes in audio focus state between the MD and the HU. Even simple user interactions (such as starting music) can cause the audio focus state to change multiple times. For example, a user listening to vehicle FM radio can connect an Android MD, launch projection mode navigation with guidance, then press the push-to-talk (PTT) button on the steering wheel to find a contact to call over Bluetooth.

The Android Auto protocol provides the appropriate projection mode requests and Receiver responses to accommodate known use cases. Integrators **MUST** use those requests and responses to design a seamless user experience across the MD and the HU.

NOTE

Audio focus is completely separate from video focus.

AAP audio sources are additions to existing audio sources. To assist the HU in prioritizing all in-vehicle audio streams, AAP uses the following audio focus negotiations:

- User **MUST** be able to view native HMI while playing audio streams from the MD.
- User **MUST** be able to view projected video content while playing audio streams from the vehicle.
- Switching audio sources does not require switching video focus.
- Switching video focus does not require switching audio focus.

AAP audio focus management is based on the [Android audio focus model](#).

Audio Focus Requests

Audio focus requests are always made from the MD to the HU. An MD sends an audio focus request to the HU when the MD wants to gain focus to play audio.

Android Auto supports four (4) types of audio focus requests:

Type	Description
GAIN	Request to gain audio focus for unknown duration. Other audio sources (i.e. vehicle audio sources) lose focus and stop playing audio. MD or the HU can have full (GAIN) audio focus.
GAIN_TRANSIENT	Request to gain audio focus for a short duration. The audio source with focus temporarily loses focus, stops playing audio, then regains focus when the MD sends the RELEASE message. The MD sends this request only when it does not already have full (GAIN) focus.
GAIN_TRANSIENT_MAY_DUCK	Request to gain audio focus for a short duration and that it is acceptable for other audio sources to keep playing after having lowered their output level (referred to as "ducking"). The MD sends this request only when it does not already have full (GAIN) focus OR GAIN_TRANSIENT focus.
RELEASE	Request to release audio focus.

The MD MUST send audio focus requests before playing audio, with exceptions for UI feedback stream audio. However, the MD MUST wait (timeout: 500ms) for a confirmed response before playing any audio, including UI feedback stream audio. If the timeout expires before the device receives a response, the device returns to its previous status.

If the HU requires time to process Audio Focus Requests (to switch audio paths, initialize amps, etc.) then the HU SHALL finish the `audioFocusRequestCallback` immediately without blocking and send its focus notification (see next section) after its required setup has completed. If setup is going to take more than 500ms, the HU SHOULD send focus notification earlier (to avoid focus request timeout) and add additional delay to audio stream by postponing ACK when MD starts sending audio stream until the setup is complete.

Audio Focus Notifications

Audio focus responses are sent by the HU to the MD to inform the MD of its focus state. These messages MAY be sent in response to audio focus requests or as unsolicited messages prompted by external events. For example, the HU sends audio focus notifications to the MD when the user selects a native (HU) audio source, such as FM radio, indicating that the MD has lost audio focus.

The audio focus responses include a valid MD audio focus state. For example, a HU response of `STATE_GAIN_TRANSIENT` to the MD is logically the same as saying the "MD has been

granted the current state of STATE_GAIN_TRANSIENT”.

Android Auto supports several audio focus states for use in audio focus notifications:

Response	Indicates ...
STATE_GAIN	MD has gained audio focus (when in response to an audio focus request) or regained audio focus (when it was temporarily lost).
STATE_GAIN_TRANSIENT	MD has gained audio focus for a short duration.
STATE_GAIN_TRANSIENT_GUIDANCE_ONLY	MD has gained audio focus for a short duration but is only allowed to play on the GUIDANCE channel.
STATE_GAIN_MEDIA_ONLY	MD has gained audio focus but is only allowed to play on the MEDIA channel.
STATE_LOSS_TRANSIENT_MAY_DUCK	MD can continue playing through MEDIA channel if volume is lowered.
STATE_LOSS	MD has lost audio focus or failed to gain audio focus.
STATE_LOSS_TRANSIENT	MD has temporarily lost audio focus or has failed to gain audio focus.

In all situations, the HU makes the final decision on audio focus. However, the HU **MUST** send audio focus notifications before playing audio to enable the MD to synchronize pause and duck timing.

The HU **SHOULD** also honor the last focus state when receiving a RELEASE notification:

- If the MD held STATE_GAIN and it relinquished audio focus (RELEASE), the HU **SHOULD NOT** automatically resume playing native music. An event other than an MD focus release, either user- or system-initiated, **SHOULD** trigger the next audio state since the MD intended to hold focus for unknown duration.
- If the MD held STATE_GAIN_TRANSIENT and it relinquished audio focus (RELEASE), the HU may immediately resume native music since the MD intended to hold focus for a short duration.

Audio Focus Initialization

After establishing a AAP connection, the HU has audio focus and the MD **MUST** request focus before playing a stream. To ensure an AAP audio implementation provides a seamless audio experience when switching from native mode to projection mode, use the following policy:

- **HU playing music.** The HU takes audio focus. If the MD is playing music, the HU denies audio focus to MD (which stops playback) and continues playing music. If user enters projection mode and starts an audio application, the HU begins switch.
- **HU not playing music.** The HU does not take audio focus. If the MD is playing music, the HU grants audio focus to the MD and continues playing music.

Audio Focus Policy

The HU and MD MUST manage audio focus according to the following policy:

1. HU MUST send focus notifications to the MD and indicate the correct focus state before playing audio from a native audio source. HU MAY NOT play native audio without previously sending a focus notification. If the focus state is unchanged, notification is unnecessary.
2. MD MUST request audio focus and indicate the desired focus state before playing audio from an MD audio source. MD cannot play audio when it does not have audio focus. MD can skip the focus request if it already has the desired focus state.
3. HU MUST grant audio focus requests from the MD in a timely manner.⁵
4. MD MUST stop audio playback if the HU is sending an audio focus loss notification (with allowances for short duration audio streaming due to asynchronous communication).
5. After granting audio focus to MD, the HU MUST play the audio stream from the MD.
6. If the audio focus notification indicates ducking is allowed, HU MAY duck audio on the media stream to allow transient audio (from MD or vehicle) on the notification stream.

Audio Focus Priorities

The HU and MD MUST manage audio stream priorities for audio focus as indicated below (priorities in descending order; lower priority numbers preempt higher numbers):

Priority	Source	Stream	Comments
1	Vehicle Notification	UI	Has priority over all other sources because it can carry safety-critical and regulatory-mandated audio notifications.
2	MD Guidance	Guidance	Has priority over all lower-priority sources. • HU MUST grant focus requests to the guidance stream unless vehicle notification audio is currently playing. • MD audio focus requests typically indicate that other streams can duck audio (see Audio Focus Requests).

⁵ Regulatory requirements may require the vehicle to reject other audio while playing regulatory notifications.

3	MD Voice	Voice	Has priority over all lower-priority sources. HU MUST grant focus requests to this stream in all cases that do not involve higher-priority streams (active guidance or vehicle notifications).
4	MD Media & Vehicle Media	Media	Has low priority. HU MUST grant focus requests to the MD media stream and stop playing vehicle media after switching focus. If a higher priority source allows ducking, these sources can pause or duck. Typically, these sources are mutually exclusive (HU cannot play audio from both sources simultaneously, ducked or otherwise).
5	MD UI Feedback	UI	Has low priority. When the MD has video focus, the HU SHOULD always play device UI feedback audio. Typically this source mixes with other audio sources without ducking.

Audio Focus Contention & Media Mixing

Many vehicles support mixing multiple audio streams, which enables an audio source to duck under a different source. If a vehicle:

- Can mix streams, the HU MAY grant STATE_GAIN or STATE_GAIN_TRANSIENT.
- Cannot mix streams, the HU MAY grant STATE_GAIN_TRANSIENT_GUIDANCE_ONLY.

Automatic Speech Recognition (ASR) Focus

ASR sessions SHOULD have exclusive focus within the car, meaning all sounds SHOULD end during a ASR session. Android Auto provides a separate VoiceSessionNotification focus status to ensure voice exclusivity:

- When the MD sends VoiceSessionRequestNotification with VOICE_SESSION_START, the HU SHOULD stop all sounds regardless of audio focus state. The MD will also request GAIN or GAIN_TRANSIENT focus to play any beep tones and the ASR response.
- When the in-vehicle automatic speech recognition is in session, the HU takes focus from MD with STATE_LOSS.

TIP

To avoid interference, the HU SHOULD lower fan speed to half (55dBA or below) when user activates voice control. When voice control ends, resume previous fan speed.

Navigation Focus

In vehicles that include a navigation system, collisions can occur between MD and native navigation guidance. Android Auto provides a separate NavFocusNotification focus status (separate from STATE_GAIN requests/responses) to prevent such collisions:

- When the MD sends NavFocusRequestNotification with NAV_FOCUS_PROJECTED, the HU SHOULD stop any existing navigation guidance and grant NAV_FOCUS_PROJECTED by sending NavFocusResponseNotification.
- When the HU sends a NavFocusResponseNotification with NAV_FOCUS_NATIVE, the MD SHOULD stop any existing navigation guidance and transition to state NAV_FOCUS_NATIVE.

For example, a vehicle that cannot mix streams cannot grant the MD access to the GUIDANCE stream during active native navigation guidance. However, the vehicle can grant NAV_FOCUS_PROJECTED to an MD, enabling it to play navigation guidance (HU stops native guidance). For details, see [Integrating Navigation](#).

Media (Music) Focus

Playing media (such as music) always involves full focus. When the MD starts music, it sends a GAIN request and the HU replies with one of the following:

- STATE_GAIN when vehicle has no restriction, **or**
- STATE_GAIN_MEDIA_ONLY when vehicle is using guidance channel, **or**
- STATE_LOSS when vehicle is playing high priority sound after stopping native media.

When vehicle starts music, the HU sends a STATE_LOSS to stop MD music and guidance.

System Stream Support

System sound stream (UI) does not require audio focus as this sound SHOULD be played as soon as possible. Support of system stream is OPTIONAL (vehicles are not required to support this stream) but lack of support prevents users from hearing system sounds.

Audio Mixing Guidelines

Some vehicles support mixing multiple audio streams while others support handling one audio stream at a time. The vehicle's ability to mix GUIDANCE and MEDIA audio streams affects the general behavior of audio focus requests and responses in Android Auto implementations. In some situations, individual audio focus implementations on the HU will reflect special circumstances of the entire vehicle audio system.

Google expects integrators to use audio requests and audio state responses to provide a good user experience in both general and special use cases. If the integrator decides to use a single-channel audio source, the HU MUST treat this source as a singular feed and not

attempt to adjust volumes based on derived audio source.

The General Audio Notification Guidelines section contains general guidelines for audio notifications. For guidelines that differ depending on mixing capabilities, see [Guidelines For Mixing Streams](#) and [Guidelines for Non-Mixed Streams](#).

General Audio Notification Guidelines

Regardless of vehicle mixed stream support, integrators can ensure a good user experience by using the following audio focus change notification logic.

When the MD is in:

- **STATE_GAIN_TRANSIENT**, the MD has focus with intent to use audio streams briefly.
- **STATE_GAIN_MEDIA_ONLY**, the MD has focus but is restricted to streaming *only* to the MEDIA channel. The MD does not send additional focus GAIN* requests.
- **STATE_GAIN_TRANSIENT_GUIDANCE_ONLY**, the MD has focus for a short period but is restricted to streaming *only* to the GUIDANCE channel. If an MD application requires full focus, the MD MUST send a focus gain request (STATE_GAIN).
- **STATE_LOSS_TRANSIENT**, the MD does not have focus. Occurs when the HU wants to take focus from MD for short period and has sent the MD a STATE_LOSS_TRANSIENT.

Guidelines For Mixing Streams

Regardless of vehicle audio capabilities, integrators SHOULD implement the following audio focus behaviors (see also [Guidelines for Non-Mixed Streams](#)).

Action	Behavior
<i>User launches AAP</i>	MD sends RELEASE request to HU so the HU has default audio focus.
<i>MD wants to play a sound</i>	MD requests focus by sending GAIN, GAIN_TRANSIENT, or GAIN_TRANSIENT_MAY_DUCK.
<i>HU grants focus to MD</i>	HU grants STATE_GAIN or STATE_GAIN_TRANSIENT to enable the MD to send audio over both MEDIA and GUIDANCE channels. The HU SHOULD grant the MD full focus whenever possible. After the HU grants focus and the MD is in STATE_GAIN or STATE_GAIN_TRANSIENT, MD apps can play guidance over the MEDIA or the GUIDANCE streams. For navigation guidance: • If HU allows only GUIDANCE, MD plays navigation guidance on the GUIDANCE stream. • If HU allows only MEDIA, MD plays navigation guidance on the

	MEDIA stream. If HU allows both GUIDANCE and MEDIA, MD SHOULD play navigation guidance on the GUIDANCE stream instead of the MEDIA stream.
<i>Vehicle requires focus</i>	HU sends STATE_LOSS to stop sound playing from the MD. For example, if the user starts the in-vehicle radio player, the HU sends STATE_LOSS to MD and music on the device stops due to focus loss. If the MD MUST play music or other audio later, it sends a focus request at that time. Alternatively, the HU can send a STATE_LOSS_TRANSIENT, which is similar to STATE_LOSS but implies that the loss is transient (temporary); i.e. the vehicle intends to stop playing sound after a short time and the HU intends to return focus to the MD. TRANSIENT states can also notify the MD that the vehicle is about to play safety critical warning sounds, in which case the vehicle SHOULD NOT wait for the MD to stop playing sound. Finally, a STATE_LOSS_TRANSIENT notifies the MD that the MD can resume playing when HU returns focus.
<i>Anytime</i>	HU can take audio focus from the MD, interrupting MD sound.

Guidelines for Non-Mixed Streams

In vehicles that do not support mixing GUIDANCE and MEDIA streams, the MD MUST send data one audio stream at a time. Make the following adjustments to audio focus behavior:

Action	Behavior
<i>No audio playing</i>	The HU SHOULD grant STATE_GAIN or STATE_GAIN_TRANSIENT to the MD when no sound is being played on either the MD or the HU.
<i>Vehicle has focus and occupies a channel</i>	When the MD cannot play on both GUIDANCE and MEDIA streams, the HU sends a STATE_GAIN_TRANSIENT_GUIDANCE_ONLY <i>or</i> STATE_GAIN_MEDIA_ONLY notification. The HU grants STATE_GAIN_*_ONLY to the MD for the unoccupied channel to prevent interruption on the occupied channel. For example, in a vehicle playing: - Native nav guidance, HU grants STATE_GAIN_MEDIA_ONLY to the MD, enabling the MD to play on the MEDIA stream. - Native media, HU grants STATE_GAIN_TRANSIENT_GUIDANCE_ONLY to

	the MD, enabling the MD to announce nav on the GUIDANCE stream.
<i>Vehicle has focus and does not occupy a channel</i>	HU MUST grant MD focus when the vehicle no longer occupies a channel. If the HU previously sent STATE_GAIN_TRANSIENT_GUIDANCE_ONLY or STATE_GAIN_MEDIA_ONLY to the MD, and the HU has finished using the channel, the HU <i>MUST</i> restore the audio focus state to the MD. For example, after completing a native guidance instruction, the HU <i>SHOULD</i> send STATE_GAIN to the MD.
<i>MD restricted to GUIDANCE stream is limited to subset of focus requests</i>	If the MD is in a STATE_GAIN_TRANSIENT_GUIDANCE_ONLY state, the MD can send a GAIN, GAIN_TRANSIENT, GAIN_TRANSIENT_MAY_DUCK, or RELEASE request. If the HU has a channel restriction, HU may respond with STATE_GAIN_TRANSIENT_GUIDANCE_ONLY for GAIN_TRANSIENT_MAY_DUCK. The HU <i>SHOULD</i> honor the intent of the audio focus requests. GAIN and GAIN_TRANSIENT expect to have exclusive focus. If the HU was previously occupying the MEDIA channel, it <i>SHOULD</i> relinquish focus upon receiving GAIN or GAIN_TRANSIENT. A GAIN_TRANSIENT_MAY_DUCK request means the MD does not expect to have exclusive focus and could be given STATE_GAIN_GUIDANCE_ONLY. GAIN requests from MD <i>SHOULD</i> be used to indicate that the MD is taking focus for a long-duration audio playback on MEDIA.
<i>MD restricted to MEDIA stream is limited to a subset of focus requests</i>	If the MD is in a STATE_GAIN_MEDIA_ONLY state, the MD cannot send GAIN* requests to the HU since STATE_GAIN_MEDIA_ONLY is treated the same as STATE_GAIN. (STATE_GAIN requests/grants access to GUIDANCE and/or MEDIA streams, but STATE_GAIN_MEDIA_ONLY requests/grants access to the MEDIA stream only.)
<i>MD stream management</i>	In STATE_*_ONLY states, the MD manages the stream to which audio is sent based on the STATE_*_ONLY restriction (this is why GUIDANCE and MEDIA streams are general descriptions and not requirements). For example, when native navigation guidance is active and MD focus state is STATE_GAIN_MEDIA_ONLY, the MD can play short messages on the MEDIA stream.

Guidelines for Audio Focus & Phone Calls

In projection mode, phone calls (incoming and outgoing) are handled by Bluetooth. **In all cases where a phone call is detected, the HU MUST stop playing native sound and the MD MUST stop playing its own sound.** If the MD has audio focus during the call, it can play

sound through allowed channels in response to an explicit user action or by implicit permission (the same rule applies to the HU, which can play its own native sound if user has allowed). For example, the MD can play navigation guidance during a call.

If the HU cannot play any sound during a phone call (vehicle cannot mix streams) and the MD had focus before the phone call, the HU SHOULD take focus away (LOSS_TRANSIENT) to prevent the MD from playing sound. The MD can still request focus in response to a user action; however, the HU SHOULD reject MD focus requests while call is active (but restore the MD focus state when the call ends).

If HU can play sound during phone call (vehicle can mix streams), the HU does not need to change focus state when the call starts. If the MD had focus before the phone call, it MUST stop playing sound when the phone call is detected but retains audio focus, enabling it to play sound in response to a user action. If the MD did not have focus before the phone call, it MUST first request focus from the HU (which SHOULD grant it) before playing sound in response to a user action.

Focus is re-evaluated after call state change, and Android Audio manager can be still in call mode. As a result, the MD may request gain (GAIN or GAIN_TRANSIENT) during an active call. The HU should reject the MD focus request while the call is active.

Audio Focus Examples

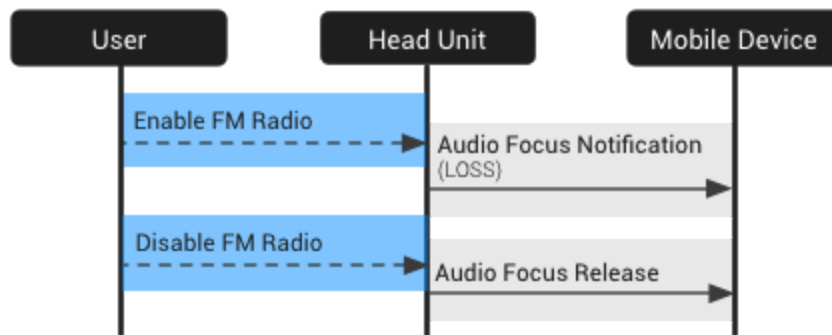


Figure 17. User enables FM Radio.

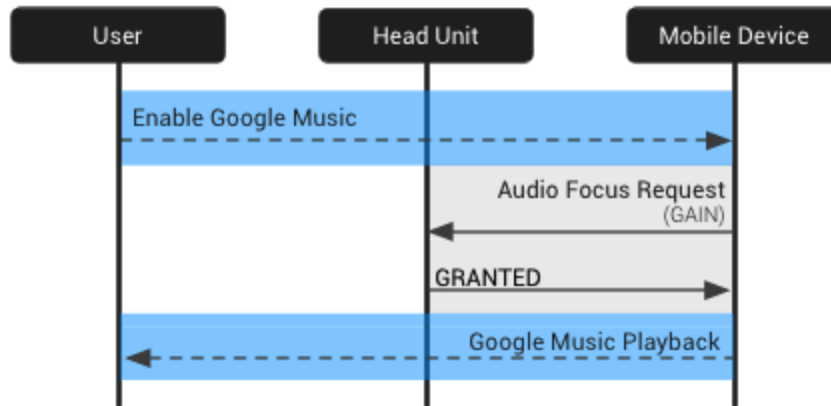


Figure 18. User enables Google Music.

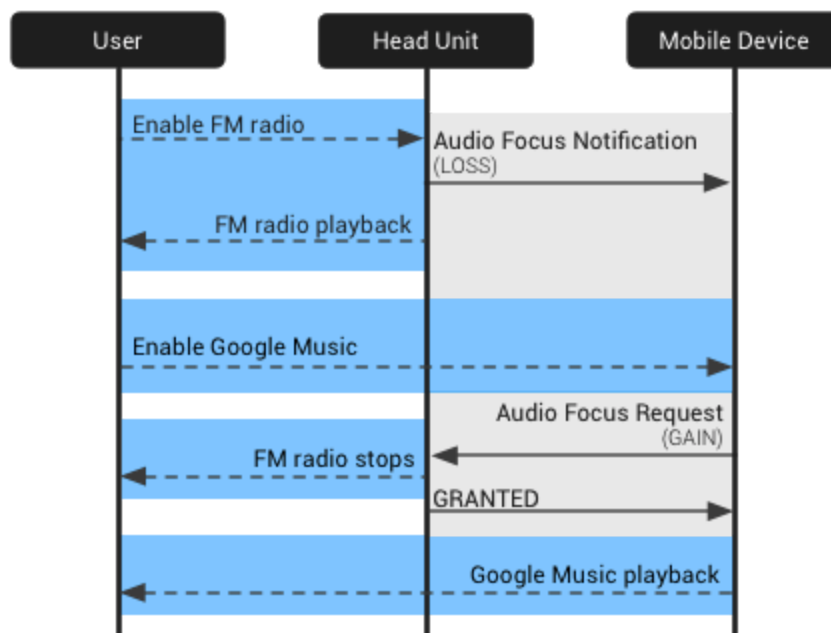


Figure 19. User enables FM Radio, then Google music.

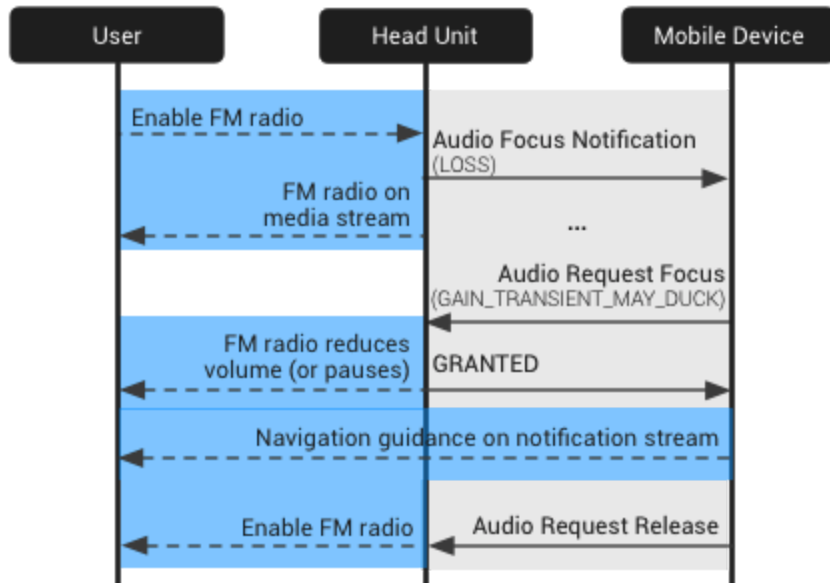


Figure 20. User enables FM Radio, then MD navigation guidance.

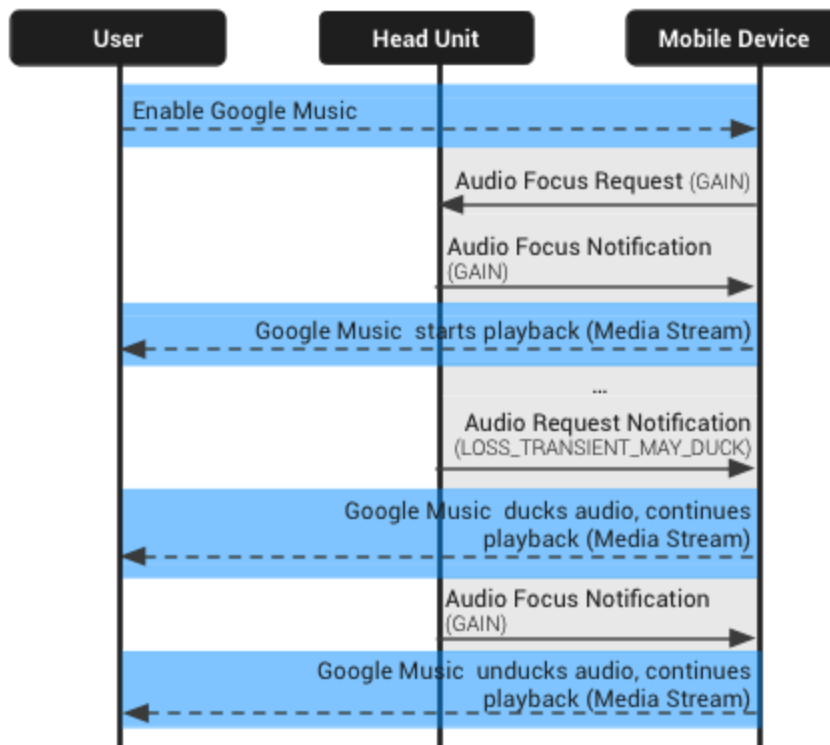


Figure 21. HU notification during Google music playback.

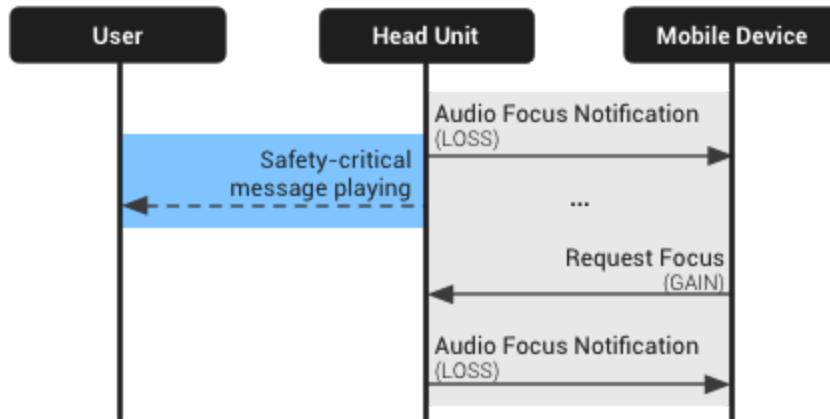


Figure 22. HU rejects audio focus request (occurs in safety or regulatory situations).

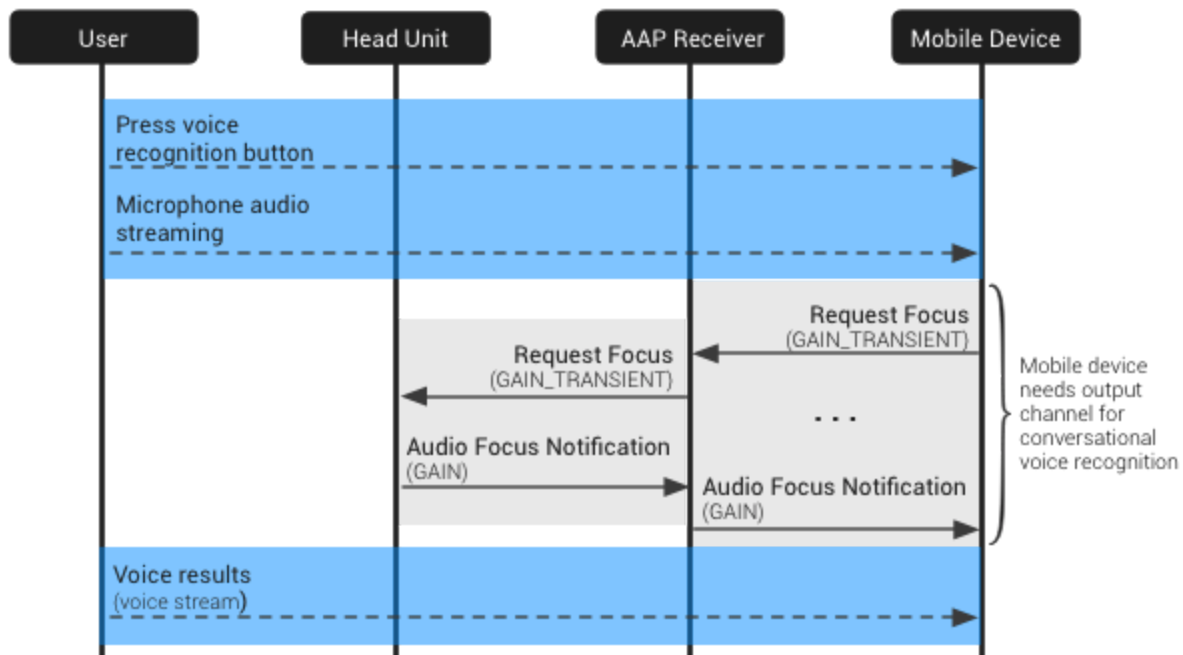


Figure 23. Automatic speech recognition (ASR).

Audio Focus Mixing vs. Non-Mixing Examples

This section includes examples of expected behavior for the following use cases:

- Native Navigation + MD Music
- MD Navigation & Music + Native FM Radio
- MD Music + MD Phone Call

Each example showcases the implementation of general guidelines, guidelines for mixing streams, and guidelines for non-mixed streams. The included call flows indicate both mobile

device (MD) requests and the vehicle/head unit (HU) responses that set audio state.

Native Navigation + MD Music

In this example, the vehicle plays native navigation guidance instructions while a user plays music via a music application on the mobile device.

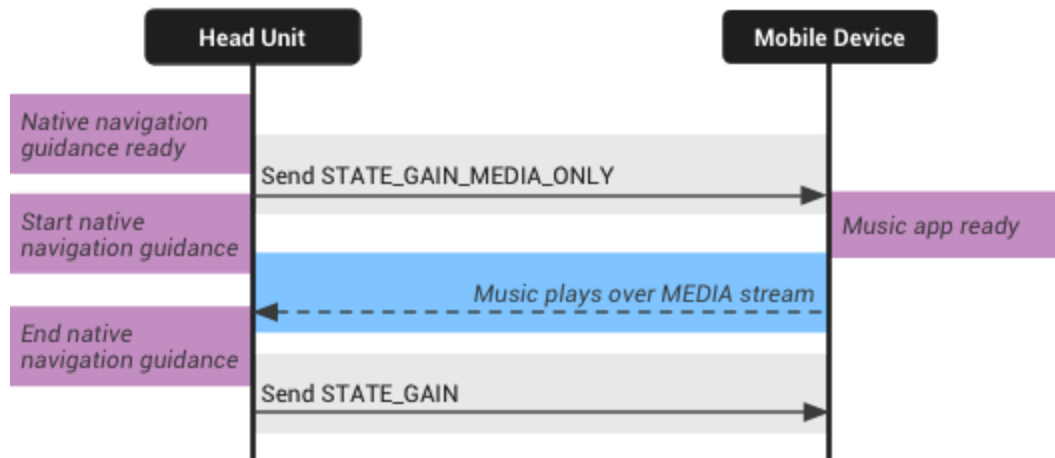


Figure 24. Native navigation + MD music, non-mixed streams.

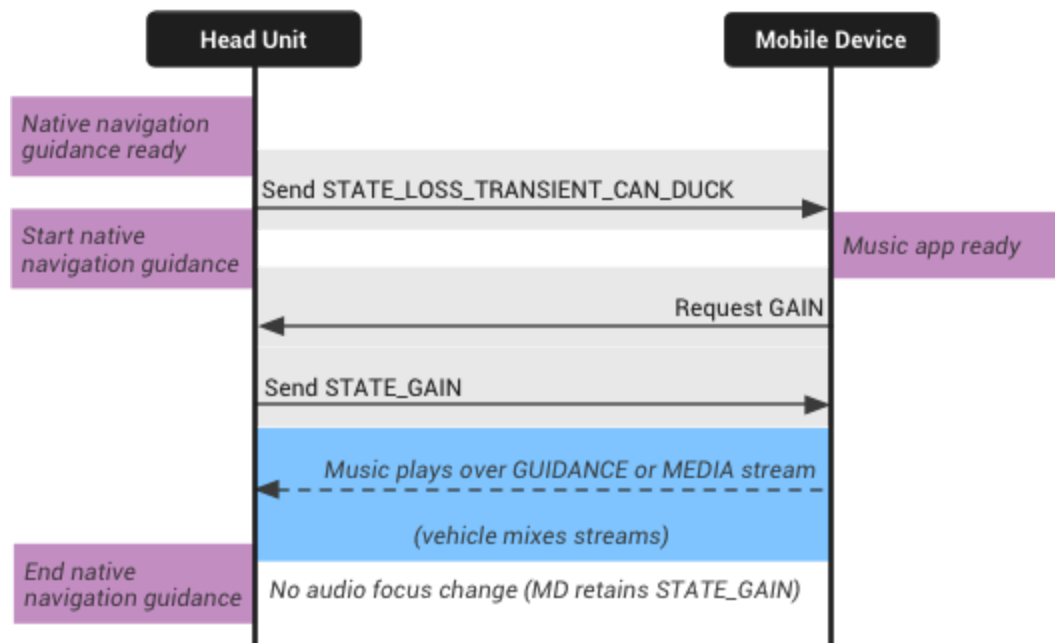


Figure 25. Native navigation + MD music, mixed streams.

MD Navigation & Music + Native FM Radio

In this example, the HU plays music from a music application on the MD while also playing

navigation guidance from the MD. The user then starts the in-vehicle FM radio, which causes the MD music application to stop playing music while continuing navigation guidance.

NOTE

If the user initiates a new audio source in the middle of an ongoing audio stream (e.g. turn-by-turn guidance), AAP truncates the audio stream.

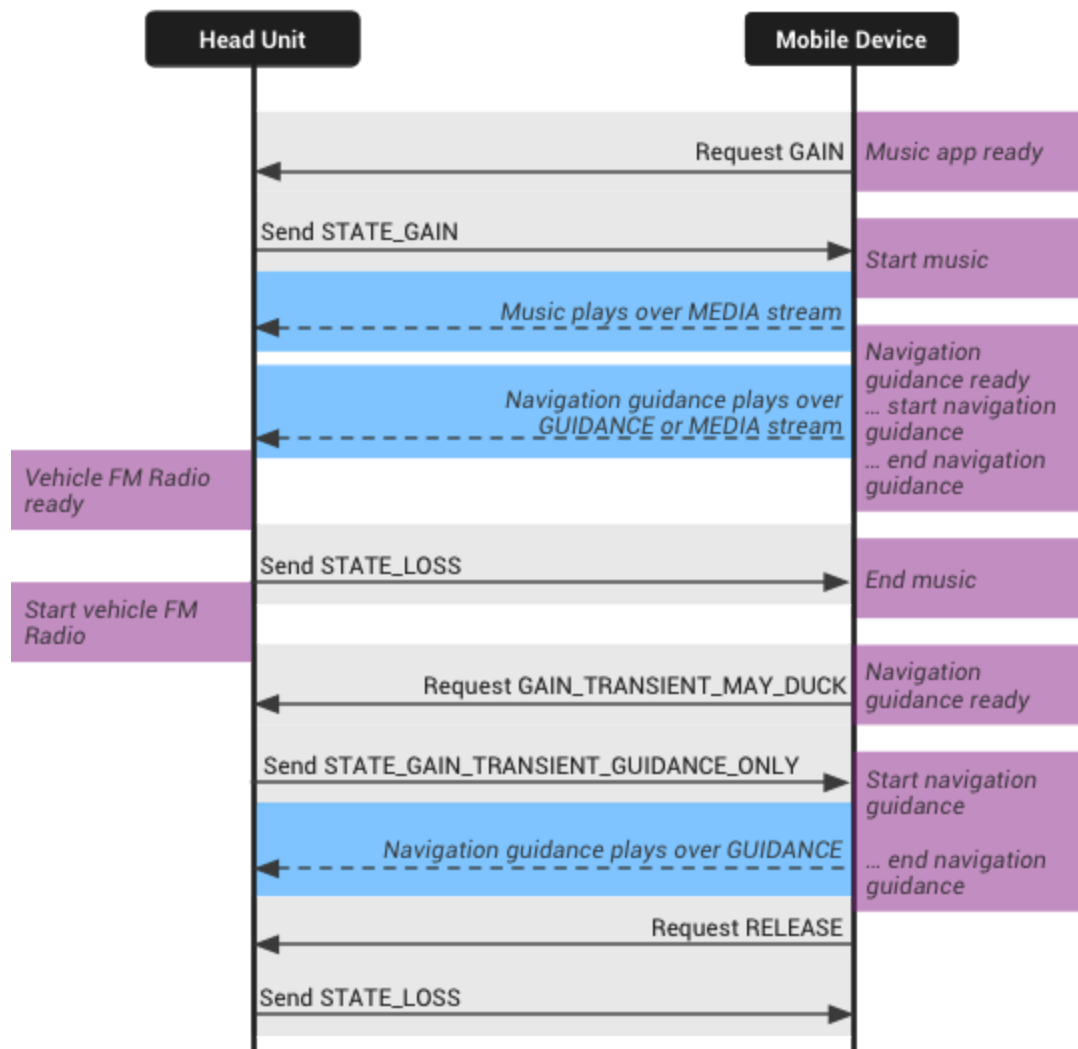


Figure 26. MD navigation/music + native FM, non-mixed streams.

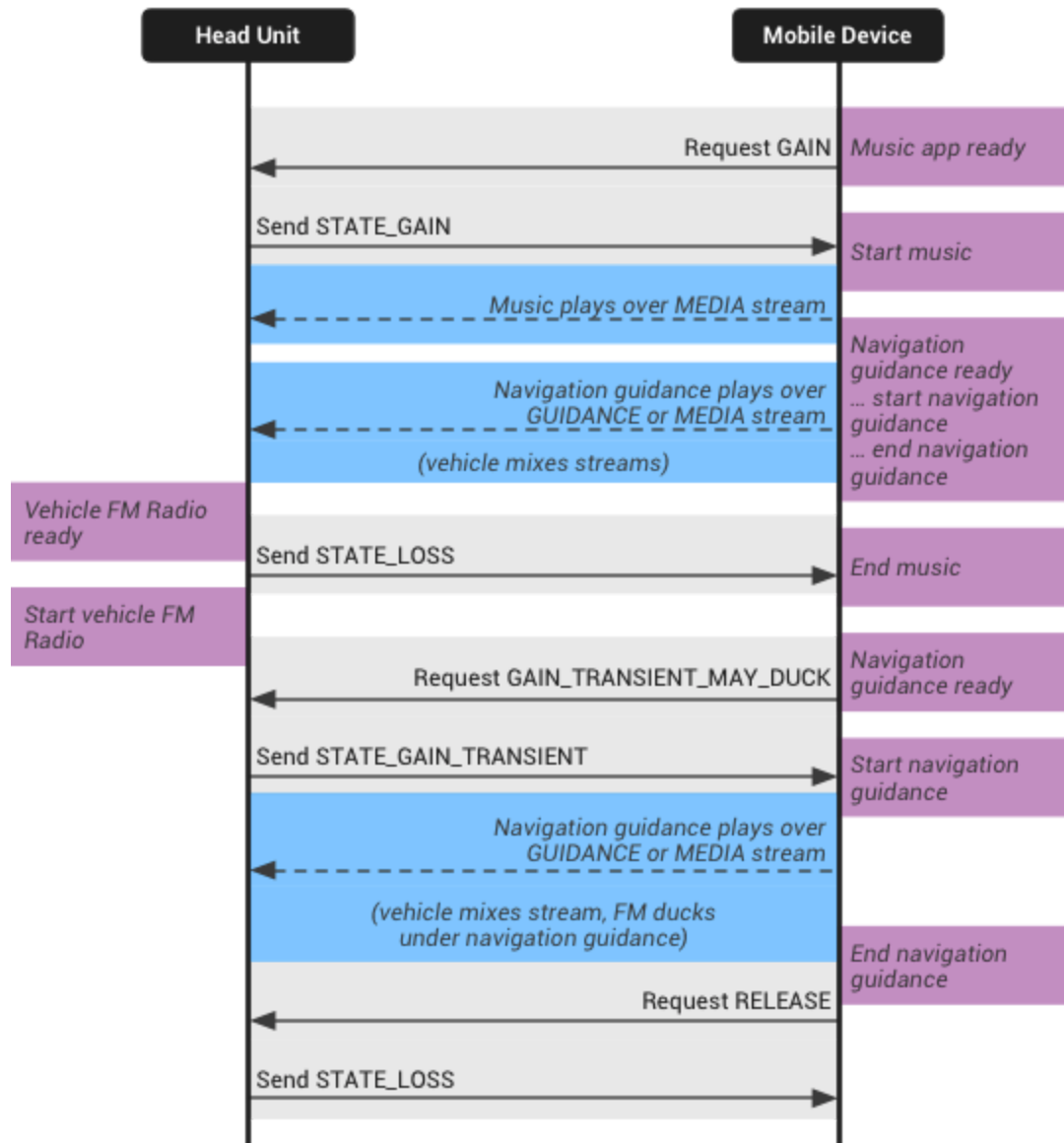


Figure 27. MD navigation/music + native FM, mixed streams.

MD Music + MD Phone Call

In this example, the user plays music via a music application on the MD. The MD receives an incoming phone call, which prompts the vehicle to stop music and play the call using Bluetooth. During the call, the user attempts to play music again via the MD music application.

In all cases (mixed streams, non-mixed streams, MD has focus, HU has focus), the music stops when the phone call is detected. During the phone call, if the vehicle:

- **Cannot mix streams**, the HU denies MD requests to start music until the call ends. If MD had focus before the incoming call, the HU sends `STATE_LOSS_TRANSIENT` focus state to notify the MD that it cannot play sound while the call is active; when call ends, the HU SHOULD restore the focus state.
- **Can mix streams**, the HU grants MD requests to start music even during active phone calls.

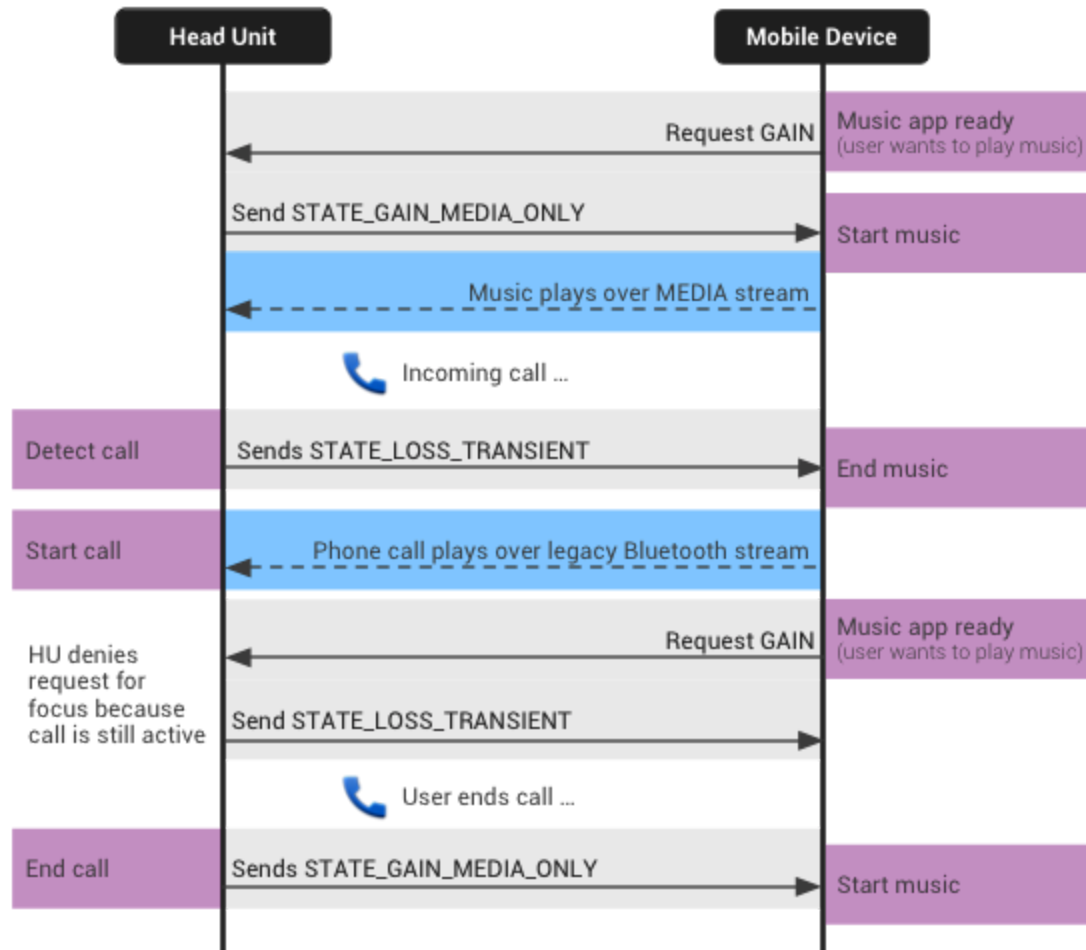


Figure 28. MD music + incoming call, non-mixed streams.

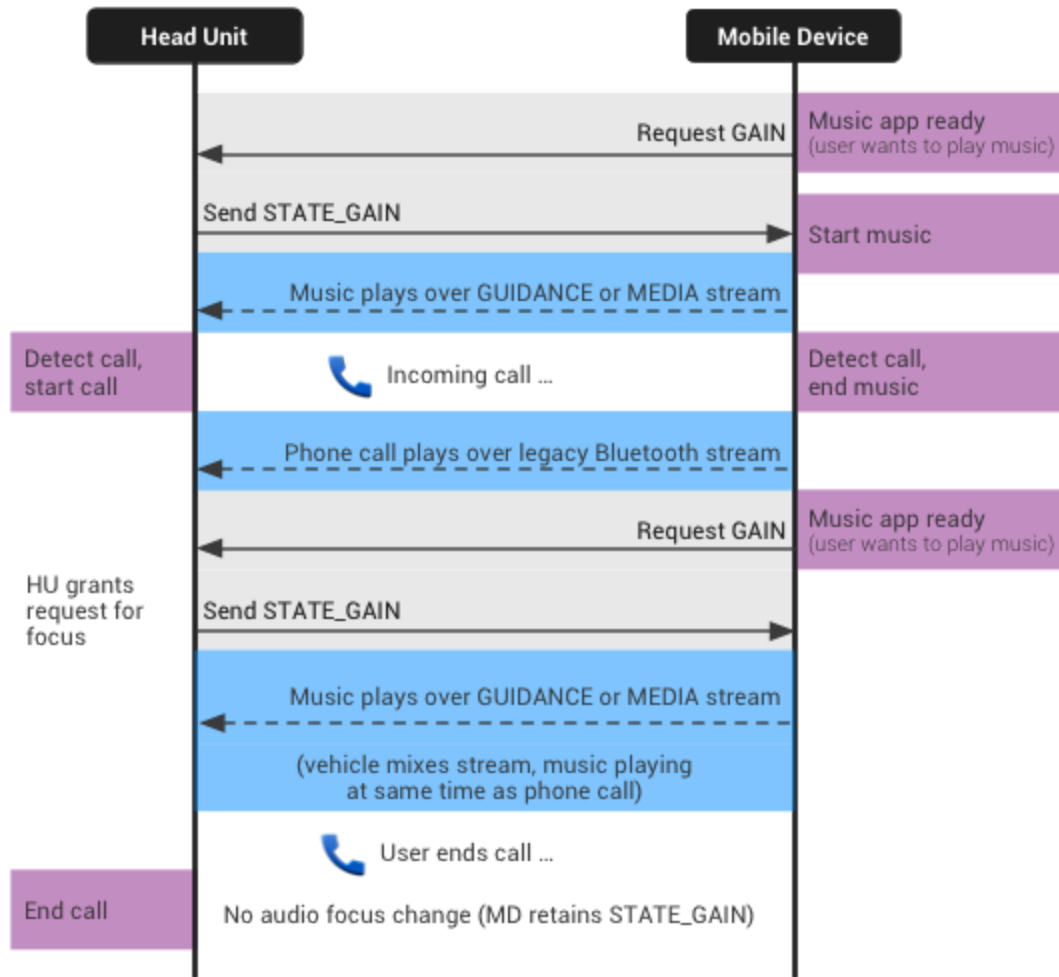


Figure 29. MD music + incoming call, mixed streams.

Latency, Flow Control & Clock Skew

AAP audio streams use an n-ACK based flow control scheme. The audio source can implement flow control using the ACK messages, including resampling in the case of clock skew. The HU SHOULD wait until it receives two frames of audio before starting playback to minimize buffer underruns.

AAP considers two (2) latencies: audio setup and audio output:

Audio setup latency (max)	500ms
Audio output latency (max)	50ms

Audio setup latency is the time required by the head unit to establish an audio stream, measured as the maximum time between an audio focus request to the head unit and a response from the head unit. This latency includes the time necessary to:

- Duck other audio
- Power Digital-to-Analog Converter (DAC) and amplifiers
- Configure head unit mixer

Audio output latency is the delay between audio being made available from the AAP receiver and reproduction of the audio on in-vehicle speakers. This latency includes:

- Buffering performed by the in-vehicle audio pipeline
- Audio codec decoder latency

Volume

All AAP audio streams are line-level volume, making the vehicle responsible for applying volume levels. The vehicle does not notify the mobile device of volume levels and the mobile device cannot request volume levels. However, the head unit can use overlay mode to display volume dialogs on top of projection UI.

Some head units support a global mute state. If the head unit is in global mute state, it can reject a mobile device focus request, but SHOULD display a message to the user that describes why the sound is not played (global mute state is enabled).

Integrating Input Devices

The head unit (HU) can pass input events from the on-board HMI to the mobile device (MD). AAP supports the following input device types:

- Touchscreens
- Buttons (including D-Pads)
- Dials/rotational controllers
- Touchpads

To provide access to an input device, OS Adaptation Layer implementations **MUST register the input device** with the AAP receiver when projection starts and **report input events** to the AAP receiver during projection for key events requested by the MD:

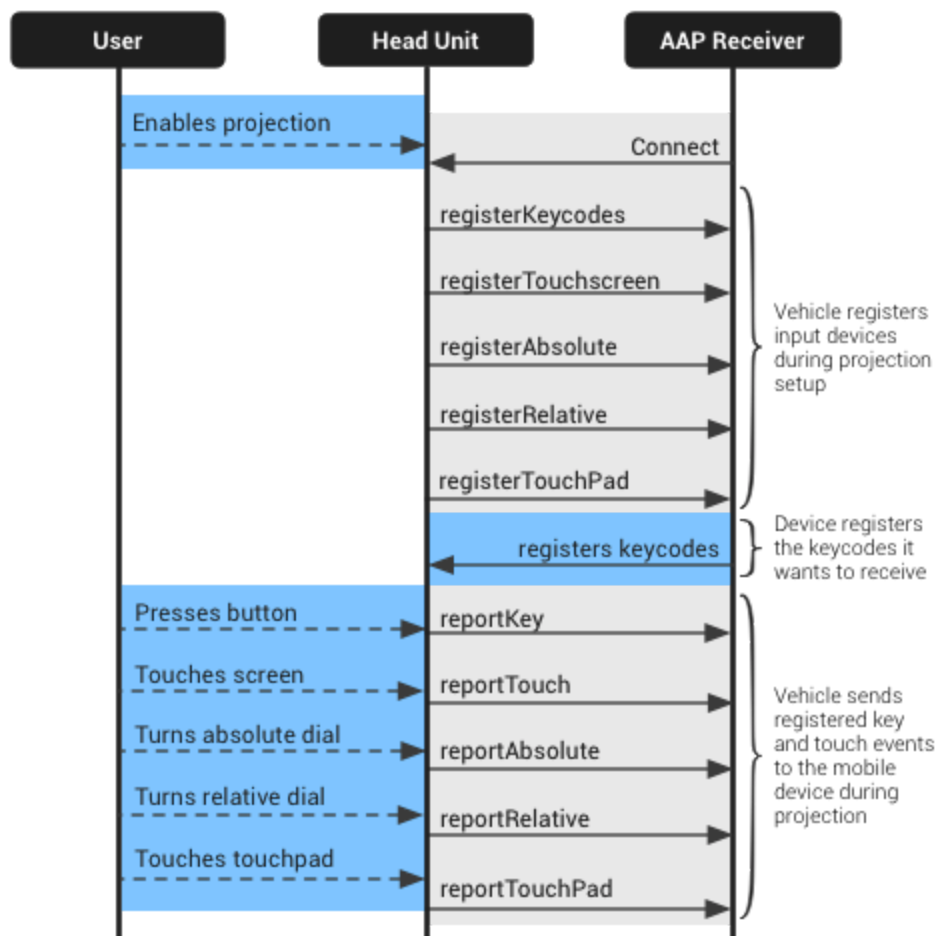


Figure 30. Input device registration and event reporting.

Touchscreen

The HU MAY pass touchscreen input events to the MD (AAP limits input events to a single touchscreen). To provide access, OS Adaptation Layer implementations MUST register the touchscreen input device with the AAP receiver and report touch events to the receiver. When video focus is Projection, the HU MUST send all touchscreen events to the AAP receiver (see [Video Focus](#)), which provides the InputSource interface that handles input device registration and event messaging (for details, refer to the [AAP receiver documentation](#)).

Buttons

Users can access the projection UI using physical buttons (including D-Pads) on the head unit (HU) HMI. To provide access to physical buttons, OS Adaptation Layer implementations MUST register each button with the AAP receiver and report key events to the receiver.

When projection mode begins, key events available to the mobile device (MD) during projection mode MUST register with the AAP receiver. However, vehicles SHOULD send key events to the receiver according to key code rules (see table below). Button events that do not impact the main console display (window defrost, hazard indicators, etc.) do not need to be sent through AAP.

Button Key Events

The vehicle sends key events to the MD through basic KeyEvent and keycodes that specify the button pressed. AAP uses the standard keycode definitions provided in the Android KeyEvent class. The OS Adaptation Layer MUST map all button events to the appropriate KeyEvent keycode before passing to the AAP receiver. Both UP and DOWN events MUST be transmitted for each keypress.

Vehicles MUST send key codes to MDs for the standard key events and focus modes:

Key Code	Focus	Description
KEYCODE_HOME	VIDEO	Causes projection UI to return to the projection home application (or launcher) from which all other projection applications are accessible.
KEYCODE_BACK	VIDEO	Handled by the application. Sending KEYCODE_BACK SHOULD NOT cause Android Auto to lose video focus. For details, refer to Android developer documentation .

KEYCODE_SEARCH	ALWAYS	Causes projection UI to open the microphone audio stream (see Audio). Sending KEYCODE_SEARCH while the microphone audio stream is already open <i>closes</i> the microphone audio stream, effectively canceling an active ASR session.
KEYCODE_CALL	ALWAYS	If a call is active on the legacy Bluetooth HFP channel, the ANSWER button SHOULD be handled by HU's HFP application (generating call control AT commands as needed on the HFP channel). At other times, the ANSWER button SHOULD send this keycode via AAP.
KEYCODE_ENDCALL	ALWAYS ⁶	If a call is active on the legacy Bluetooth HFP channel, the END button SHOULD be handled by HU's HFP application (generating call control AT commands as needed on the HFP channel). At other times, the END button SHOULD send this keycode via AAP.
KEYCODE_MEDIA_NEXT	AUDIO	Indicates selected media SHOULD advance to next track or station.
KEYCODE_MEDIA_PLAY	AUDIO	Indicates selected media SHOULD play.
KEYCODE_MEDIA_PLAY_PAUSE	AUDIO	Indicates selected media SHOULD play or pause (toggle).
KEYCODE_MEDIA_PAUSE	AUDIO	Indicates selected media SHOULD pause.
KEYCODE_MEDIA_PREVIOUS	AUDIO	Indicates selected media SHOULD return to previous track or station.
KEYCODE_DPAD_UP	VIDEO	Sends DPAD_UP event to projected UI.
KEYCODE_DPAD_DOWN	VIDEO	Sends DPAD_DOWN event to projected UI.
KEYCODE_DPAD_LEFT	VIDEO	Sends DPAD_LEFT event to projected UI.

⁶ Send keys to the mobile device unless the integration uses legacy Bluetooth connection for telephony audio.

KEYCODE_DPAD_RIGHT	VIDEO	Sends DPAD_RIGHT event to projected UI.
KEYCODE_DPAD_UP_LEFT [†]	VIDEO	Sends DPAD_UP_LEFT event to projected UI.
KEYCODE_DPAD_DOWN_LEFT [†]	VIDEO	Sends DPAD_DOWN_LEFT event to projected UI.
KEYCODE_DPAD_UP_RIGHT [†]	VIDEO	Sends DPAD_UP_RIGHT event to projected UI.
KEYCODE_DPAD_DOWN_RIGHT [†]	VIDEO	Sends DPAD_DOWN_RIGHT event to projected UI.
KEYCODE_ROTARY_CONTROLLER [†]	N/A	Special keycode used as a device identifier and reported during initialization (not sent as normal keycode). Rotary controller events are sent using relativeEvent api calls. The delta value SHOULD be positive for clockwise rotation and SHOULD be negative for counterclockwise rotation. An absolute value of one corresponds to a single step in the UI and scales linearly for larger absolute values.
KEYCODE_MEDIA [†]	VIDEO [‡]	Causes projected UI to switch to media screen.
KEYCODE_NAVIGATION [†]	VIDEO [‡]	Causes projected UI to switch to navigation screen.
KEYCODE_TEL [†]	VIDEO [‡]	Causes projected UI to switch to telephony screen.
KEYCODE_PRIMARY_BUTTON [†]	N/A	Special keycode used as a device identifier and reported during initialization. Button events are sent using absoluteEvent api calls. The event value SHOULD be 1 when the button is pressed and 0 when the button is released.
KEYCODE_SECONDARY_BUTTON [†]	N/A	Special keycode used as a device identifier and reported during initialization. Button events are sent using absoluteEvent api calls. The event value SHOULD be 1 when

		the button is pressed and 0 when the button is released.
KEYCODE_TERTIARY_BUTTON†	N/A	Special keycode used as a device identifier and reported during initialization. Button events are sent using absoluteEvent api calls. The event value SHOULD be 1 when the button is pressed and 0 when the button is released.

† Defined in the Android Auto protocol specification.

‡ Keycodes that cause the projected UI to switch to a different screen (KEYCODE_MEDIA, KEYCODE_NAVIGATION, KEYCODE_TEL) SHOULD also cause Android Auto to request video focus.

Implementing Button Interface

The AAP receiver provides the InputSource interface that handles input device registration and event messaging. For details on InputSource, refer to the [AAP receiver documentation](#).

Rotational Controller Devices

AAP supports rotational input devices—typically physical dials associated with the main head unit. To provide access to these devices, OS Adaptation Layer implementations MUST register each device with the AAP receiver and report input events to the receiver. When projection mode begins, all absolute and relative value device events that impact the main console display MUST register with the AAP receiver.

Dial Key Codes

Motion of a rotary controller is modeled as a series of keycodes, with a keypress being sent for each step that the controller is moved. For details, see [Button Key Codes](#).

Implementing Device Interface

The AAP receiver provides the InputSource interface that handles input device registration and event messaging for all input device types (for details, refer to [AAP receiver documentation](#)).

Touchpad

AAP supports touchpad input devices both as a primary user interface for navigating the UI, and as a secondary device used for handwritten text input and map scrolling. To provide access to these devices, OS Adaptation Layer implementations MUST register each device with the AAP receiver and report input events to the receiver. Touchpads MUST be able to send absolute position values, and values for multiple fingers can be sent.

Implementing Device Interface

The AAP receiver provides the InputSource interface that handles input device registration and event messaging for all input device types. For details on InputSource, refer to the [AAP receiver documentation](#).

Devices that support haptic feedback SHOULD register to receive feedback events from the MD. OEMs that wish to implement haptic feedback SHOULD work closely with Google to determine hardware integration specifics.

Integrating Automatic Speech Recognition

AAP relies on Automatic Speech Recognition (ASR) for safe and convenient use of AAP while driving. Voice is the preferred method of in-vehicle interaction and requires careful AAP integration. When users launch ASR, the system uses the vehicle cabin microphones and passes audio to the mobile device (MD) via the infotainment head unit (HU) and the USB AAP link. ASR is assumed to be conversational, involving parallel audio input and audio output.

Initiating Automatic Speech Recognition

AAP supports the following ASR initiators:

Method	Description
<i>Hard Button</i>	Triggers the active ASR system. Vehicles MUST have a hardware button to support push-to-talk (PTT). • The button SHOULD be a dedicated ASR button that is accessible by the driver. • Alternatively, implementors MAY assign ASR to the call button.
<i>Mobile Device Request</i>	MD initiates off-board ASR for specific user actions (such as user pressing a soft voice button while in projection mode).

Typically, a vehicle with Android Auto has two ASR systems:

- **Mobile Device** (*off-board* or MD ASR). Devices have access to powerful ASR engines, either on the device or backed by cloud services.
- **Head Unit** (*on-board* or HU ASR). Vehicles have on-board ASR for use cases involving native functionality such as "Turn on radio". Support continues during active AAP connections.

Users can issue voice commands to the ASR system of choice by pressing the voice input (push-to-talk/PTT) button. Android auto supports two (2) methods for initializing an ASR session: short-press and long-press.

Short-Press

This implementation relies on the HU sending the TTS key command as soon as the PTT button is pressed down. The HU SHOULD trigger the ASR experience that corresponds to the current video focus state, i.e., trigger MD ASR if Android Auto has video focus (send the KEYCODE_SEARCH) or trigger the HU ASR if Android Auto does not have video focus.

The short-press approach is STRONGLY RECOMMENDED.

Long-Press

If the HU reserves short-press for native ASR, a timer based long-press (500ms) will activate Android Auto ASR regardless of video focus.

Voice Recognition Session

AAP ASR occurs within a Voice Recognition (VR) session, which is bounded by matching `voiceSessionNotificationCallbacks` from the MD:

- An asynchronous `voiceSessionNotificationCallback` is sent on the registered controller callback when the VR session starts.
- A second `voiceSessionNotificationCallback` is sent when the VR session is completed.

Cancelling or Restarting a VR Session

During an active AAP VR session, short-press of the ASR hard button SHALL cancel the active session, and long-press SHALL restart the AAP VR session. For vehicles that have native ASR and utilize short-press to initiate a VR session, when an AAP VR is in session:

- Initial short-press MUST cancel an active AAP VR session, and
- Subsequent short-press MUST trigger the native ASR session.
- Single short-press MAY NOT trigger both a VR Cancel and VR Start.
- HU MUST NOT send successive short-press commands to stop and start the session.

Implementing Voice Recognition (VR) Actions

AAP requires a hardware button to trigger ASR. You can set the button to trigger AA with **short-press** (preferred) or **long-press** (and leave short-press for native VR/fallback).

A hard button affects the AA VR state using **Start VR**, **Restart VR**, and **Cancel VR** actions. The `reportKey` and `reportKeyLongPress` functions send button presses to the MD to trigger those actions. When calling `reportKey` or `reportKeyLongPress`, pass the keycode for `KEYCODE_SEARCH`. For details, refer to **InputSource.h** in Receiver Library Documentation.

Short-Press Standard Session

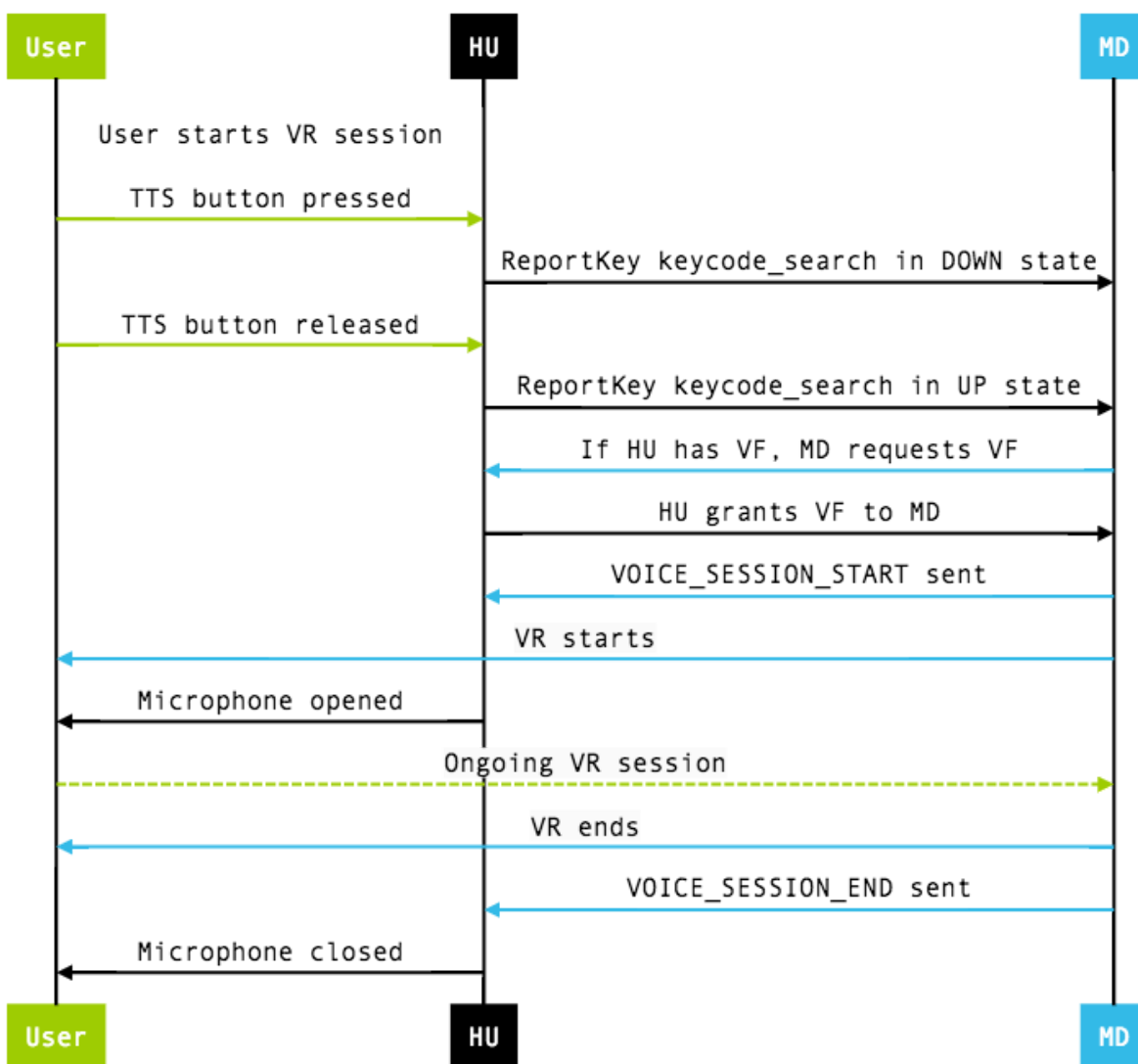


Figure 31. Short-press, standard implementation, standard session

Short-Press Restart

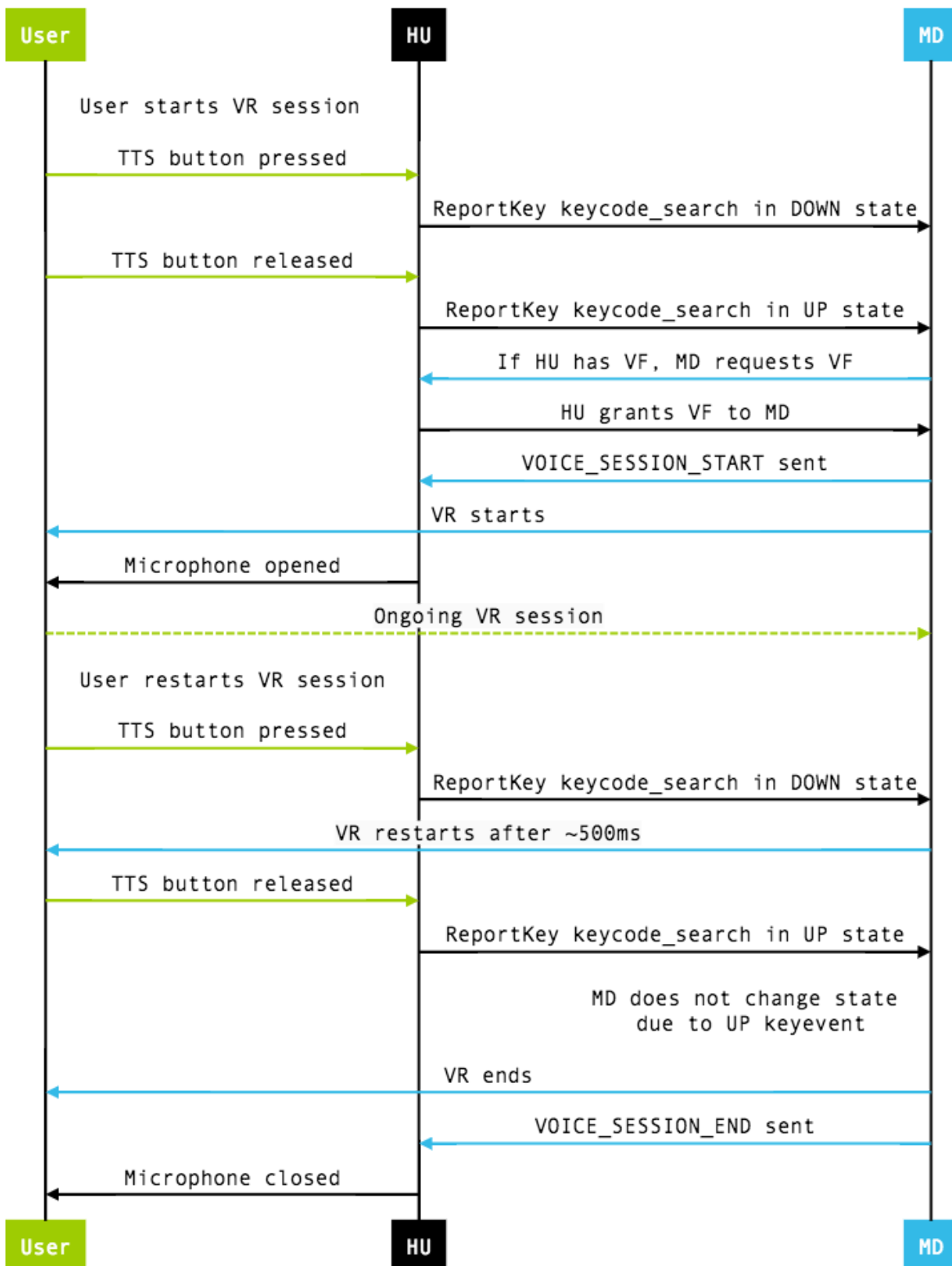


Figure 32. Short-press, standard implementation, restart

Short-Press Cancel

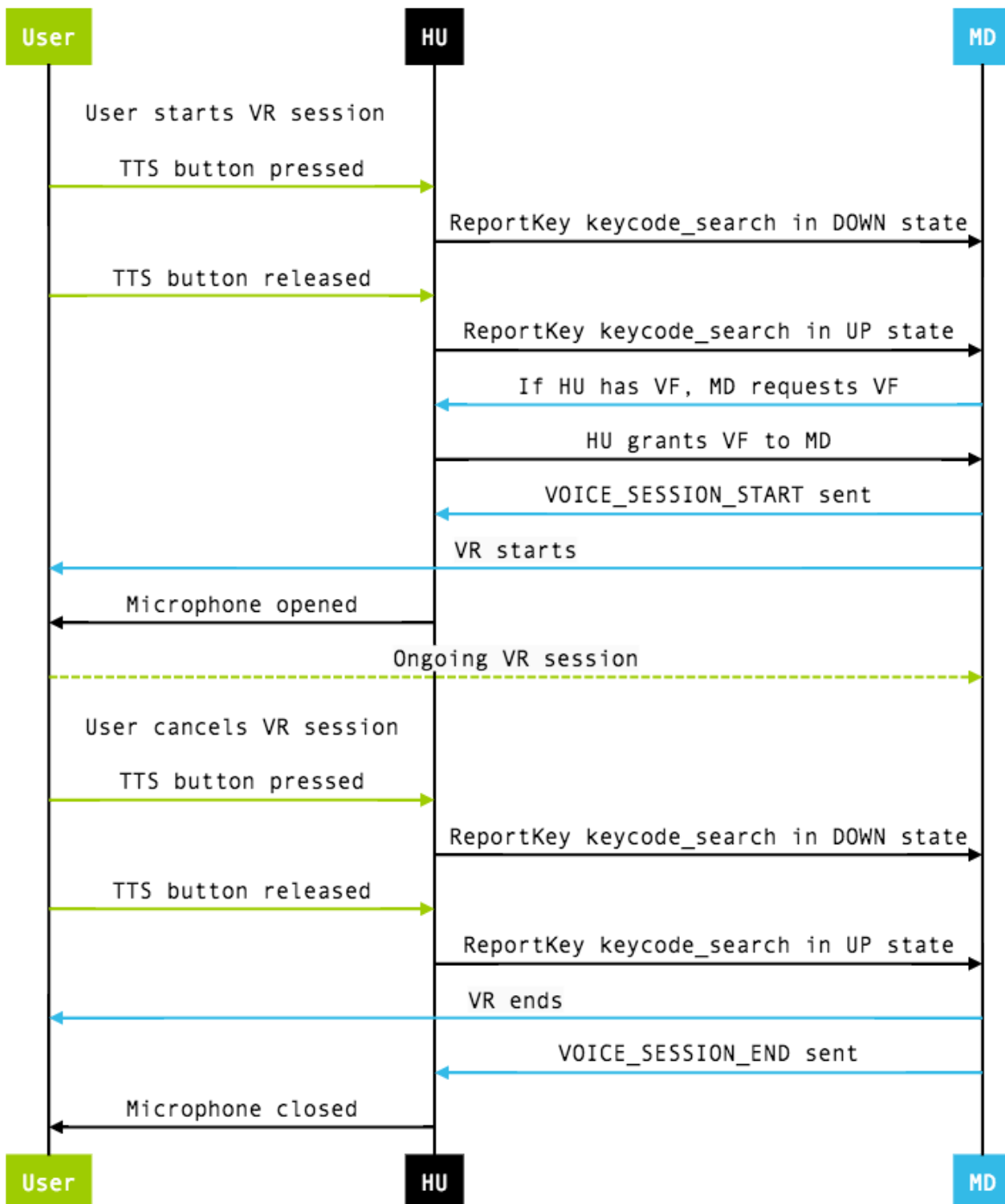


Figure 33. Short-press, standard implementation, cancel

Long-Press Standard Session

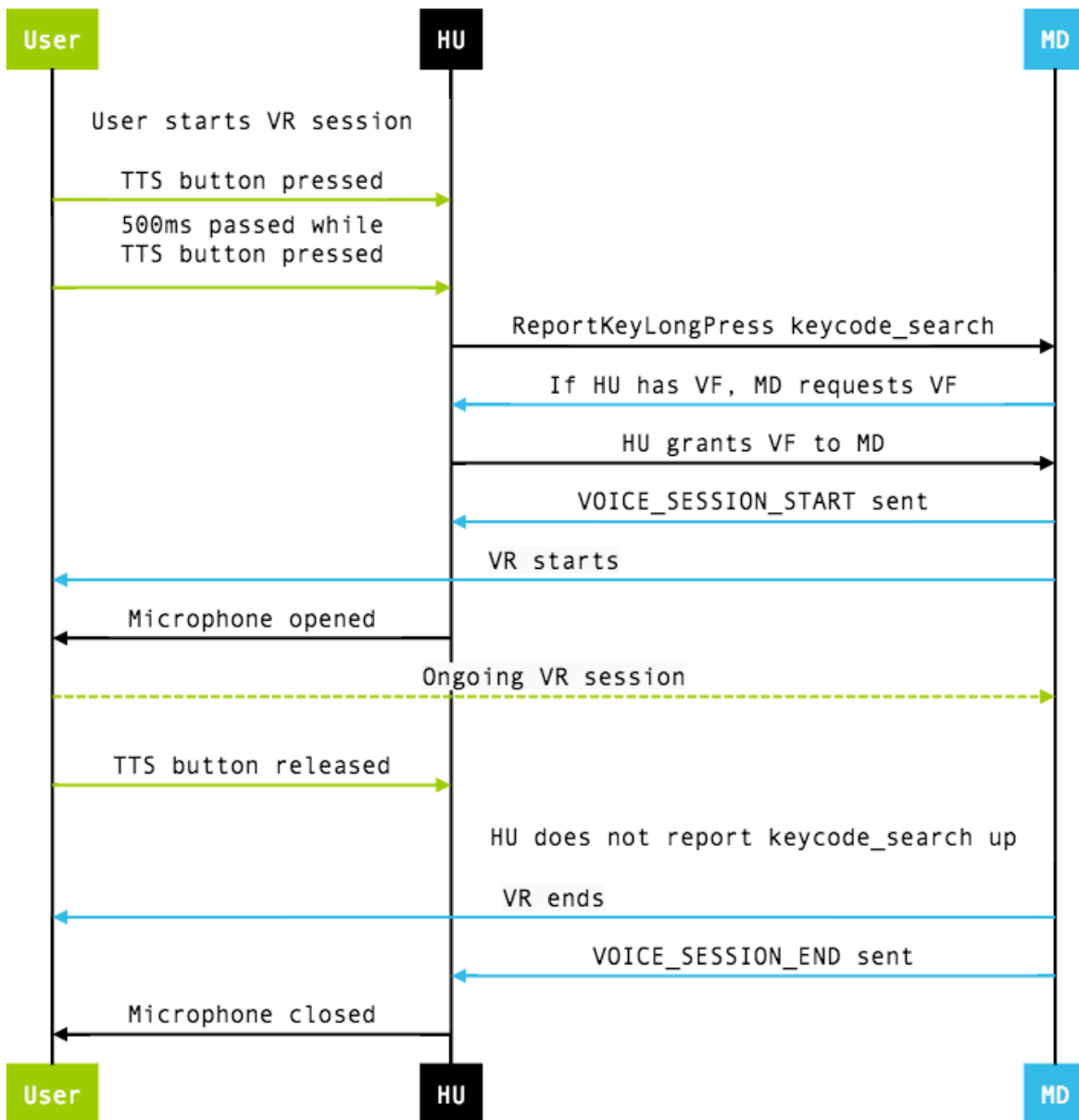


Figure 34. Long-press, standard session

Long-Press Restart

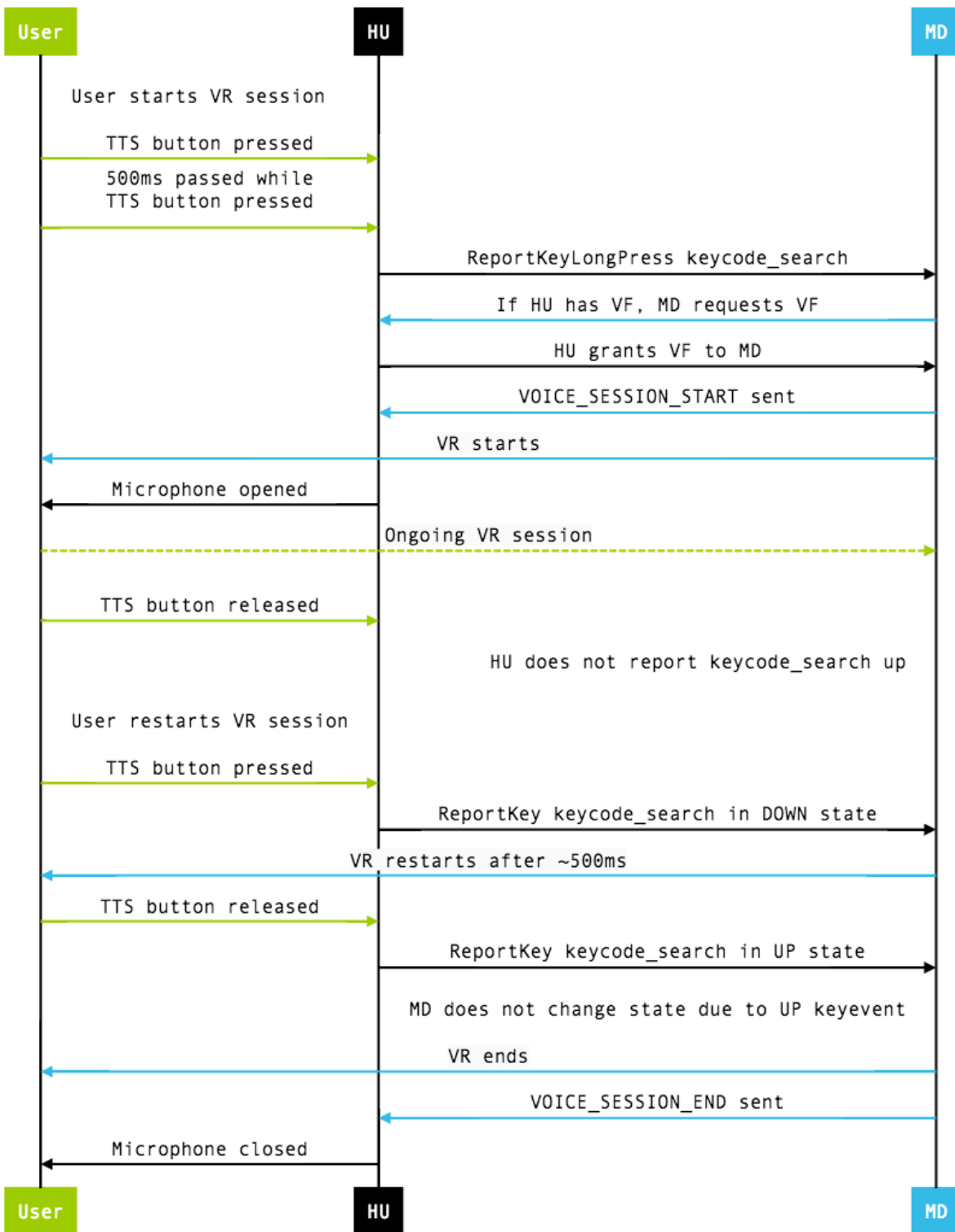


Figure 35. Long-press, restart

Long-Press Cancel

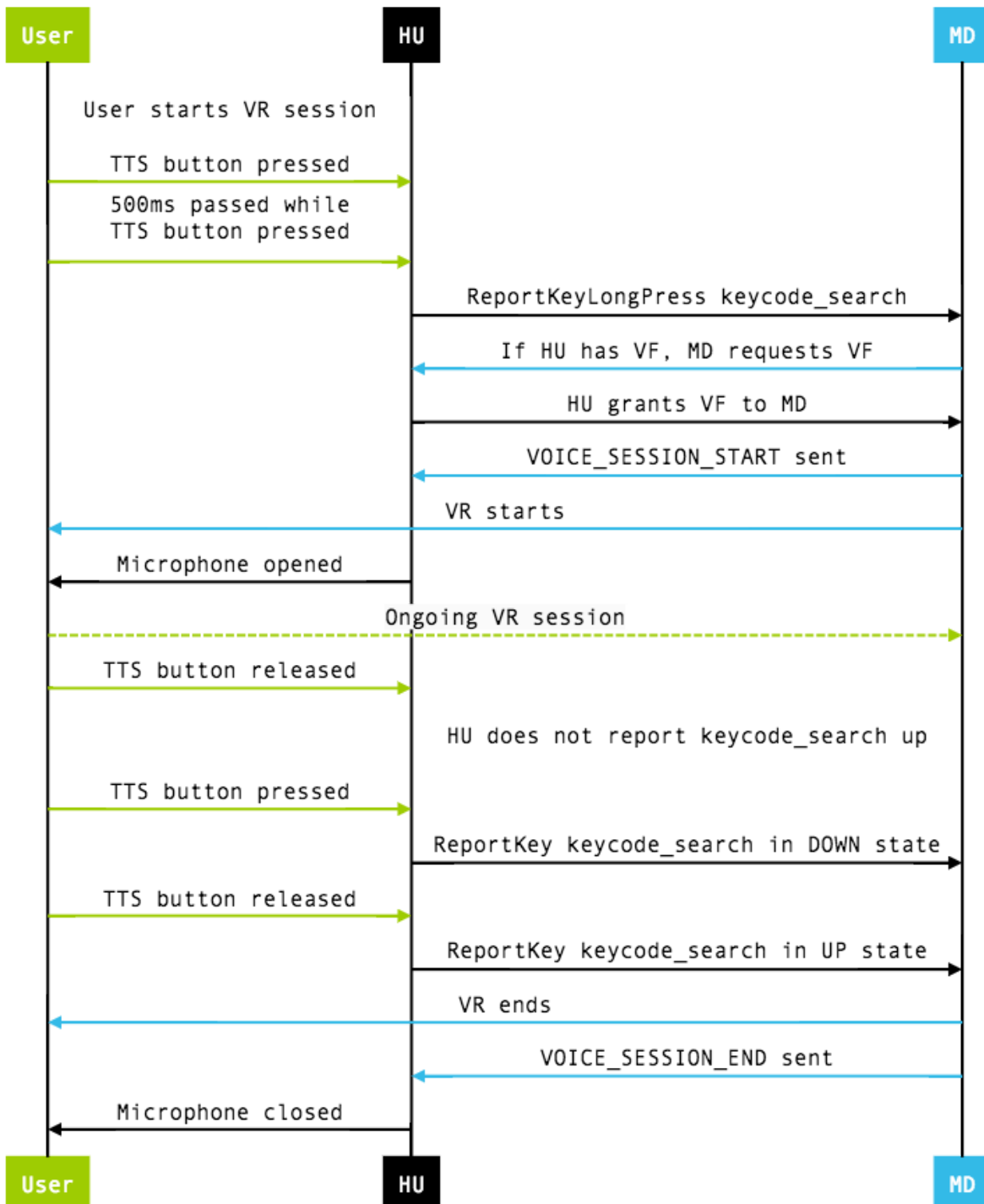


Figure 36. Long-press, cancel

Speech Audio Format

To ensure proper functionality, the microphone and HU MUST support the following audio format and audio processing requirements:

Format	PCM
Sampling Rate	16KHz
Sample Size	16Bits
Buffer Size	2048 (frames MUST be multiples of Buffer Size)
File Type	.wav
Stereo Type	Mono

Processing Requirements

The following processing settings are REQUIRED:

- **Noise Reduction Processing.** MUST be disabled for single-mic systems. MAY be enabled for multiple-mic systems that use techniques such as beamforming to improve results in adverse conditions.
- **Automatic Gain Control (AGC).** MUST be disabled. Google Automatic Speech Recognition (ASR) uses its own AGC to adjust for audio volume. Interactions between third-party and Google AGC are generally poor, leading to reduced overall quality.

Processing Recommendations

The following processing settings are RECOMMENDED:

- **Amplitude.** Device SHOULD exhibit flat amplitude versus frequency characteristics (+- 3 dB 150 Hz to 7000 Hz).
- **Audio Input Sensitivity.** Set such that 90 dB SPL at 1000 Hz yields RMS of 2500 for 16 bit samples.
- **Harmonic Distortion.** Total harmonic distortion SHOULD be less than 1% from 100 Hz to 4000 Hz at 90 dB SPL input level.
- **High Pass Filter.** High-pass filter to attenuate sensitivity to frequencies below 20 Hz by at least 30 dB.

Integrating Vehicle Data

When connected to a AAP mobile device (MD), the vehicle head unit (HU) MUST send read-only vehicle information to the MD using a pub-sub model: The MD registers for specific events and indicates an update frequency. On initial registration from an MD, the vehicle MUST notify the device of the current value for an event (if data is available).

NOTE

This section describes vehicle data exposed to the platform, but not necessarily to platform applications. In all cases, access to the vehicle data for mobile device applications is governed using the Android Permission model.

If a vehicle makes a type of vehicle data available via AAP, the implementation MUST always make the data available, regardless of the current video and audio focus state. Vehicle data MUST be shared as specified below, and MUST never be faked, simulated, or falsified. Sensor errors can be reported to MD using the SensorError message when identified by the HU.

AAP supports the following required and OPTIONAL vehicle data sources.

Driving Status

Vehicle Data Type	Required	Min Rate	Details
<i>RPM</i>	OPTIONAL	10 Hz	
<i>Current Gear</i>	OPTIONAL	on change	NEUTRAL, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, DRIVE, PARK, REVERSE. This is an OPTIONAL status message. If present, it MUST accurately represent the actual transmission/gear status and MUST NOT be synthesized. If the actual transmission/gear status is not available, this status SHOULD NOT be sent. Automatic: transmission state (generally NEUTRAL, DRIVE, PARK, REVERSE). Manual: transmission state including gear selection.
<i>Odometer</i>	OPTIONAL	on change	Resolution: 100 meters
<i>Parking Brake</i>	OPTIONAL	on change	

Fuel

Vehicle Data Type	Required	Min Rate	Details
<i>Estimated Range</i>	OPTIONAL	on change	Estimated vehicle range remaining to nearest 1km; same data as shown in other displays (e.g. instrument cluster). Conveys % fuel remaining.
<i>Fuel Level Low</i>	OPTIONAL	on change	Low fuel indicator

Navigation

If the HU has navigation sensors, those sensors **MUST be provided over AAP**. If the HU does not have navigation sensors or those sensors don't comply with minimum requirements, you MUST apply for and be granted a waiver by Google, as well as pass the Projection Compatibility Test Suite (PCTS). For example, Google may grant a waiver to a vehicle that does not have GPS if the vehicle passes other PCTS tests that ensure the position of the MD allows for GPS positioning from the device.

Vehicle Data Type	Min Rate	Details
<i>GPS Location</i>	Use the same update rate as the underlying GPS hardware (typically 1 Hz).	Position of the vehicle, usually from a GPS source. Uses WGS84 datum. May include corrections from sensor fusion, but MUST NOT include Map Matching (MM). • Latitude & Longitude • Accuracy (68% confidence horizontal radius) • Altitude • Speed (as calculated by Location engine) • Course (as calculated by Location engine⁷) The HU SHOULD send raw GPS values to the MD. Dead Reckoning (DR) and/or sensor fusion may be applied only if the HU reports the transformations in the locationCharacterization flags. MM MUST NOT be applied on the HU as it will interfere with the MD's own MM. During initial sensor set up, the HU SHOULD report what transformations it performs on GPS data

⁷ In this context, "Location engine" can be interpreted as the GPS module.

		using locationCharacterization flags. If no GPS fix is available and the number of reported GPS satellites is zero, the HU MAY send location data to the MD.
<i>GPS satellite status</i>	Use the same update rate as the underlying GPS hardware (typically 1 Hz).	The number of GPS satellites used in determining the location (REQUIRED), and the number of satellites in view (OPTIONAL). This data MUST be sent regardless of whether location fix was acquired or not (i.e. no silence periods). If locations are ever provided when no GPS signal is available (through dead-reckoning), it is imperative that the number of satellites in the fix (zero) is supplied. Per-satellite information is OPTIONAL.
<i>Mechanical Speed</i>	Use same rate as underlying hardware. Minimum rate 1 Hz.	m/s. Derived from mechanical rotation at transmission or wheels (not GPS). MUST be a negative value when vehicle is moving in reverse.
<i>Compass Heading</i>	Use same rate as underlying hardware. Minimum rate 1 Hz.	1-axis compass heading (or OPTIONAL 3-axis if pitch and/or roll are available). Compass heading updates differ from the course in a location update because the heading: • Can be updated independently of location information, enabling compass updates when no accurate location is available (such as in tunnels). • Always indicates the direction of the front of the vehicle, whereas location course indicates the direction of movement (velocity vector). These are usually the same when travelling forwards, but are 180 degrees apart when reversing. The compass heading is the orientation of the vehicle as determined by magnetic sensors; location course is the direction of movement (velocity vector) as determined by the location engine. If the HU does not have access to a compass heading that is independent of the

		location course, then compass updates SHOULD NOT be sent. The compass heading SHOULD be magnetic north (rather than true north). The RECOMMENDED resolution of compass heading data is 1 degree, and the minimum resolution is 45 degrees. As compass data can be noisy, gyro stabilisation is acceptable but MUST NOT be a duplicate of GPS bearing data.
<i>Accelerometer</i>	Use same rate as underlying hardware. Minimum rate 3 Hz.	3-axis acceleration measurements. Used for Dead Reckoning on the MD. For details, refer to Motion Sensors, Figure 1 , where positive X axis points to the right door, positive Y axis point to the nose of the car, and positive Z axis points to the roof of the car.
<i>Gyroscope</i>	Use same rate as underlying hardware. Minimum rate 3 Hz.	Up to 3-axis gyroscope measurements. Z axis (left-right turns) is REQUIRED; other axes are OPTIONAL but encouraged. Used for Dead Reckoning on the MD. For details, refer to Motion Sensors, Figure 1 .

If the vehicle does not have GPS, the MD GPS is used instead. Such vehicles MUST provide a location for the device such that the performance of the MD GPS is equivalent to performance outside of the vehicle.

Privacy

Vehicle Data Type	Required	Min Rate	Details
<i>Passenger Presence</i>	OPTIONAL	on change	Google services are a personalized experience, so it is important to know if the passenger seat is occupied before projecting video or audio content you may not want a passenger to hear.

Safety

Vehicle Data Type	Required	Min Rate	Details
<i>Drive Level</i>	REQUIRED	on change	Indicates the level of feature/functionality lockout as determined by vehicle. Defined as a bitmask: 00000 = Unrestricted. 00001 = No video playback allowed. Video playback is media such as movies, YouTube, games, etc. (not UI). 00010 = No text input allowed (on-screen keyboard, rotary controller speller, touchpad text entry)⁸. Limits UI interaction. 00100 = No voice input allowed. 01000 = No setup/configuration allowed⁹. 10000 = Limit displayed message length. OEMs MUST select drive level in compliance with the laws and regulations that apply to markets where the product is shipping.
<i>Day/Night Mode</i>	REQUIRED	on change	Indicates use of night mode when rendering the UI. Defined as a boolean: 1 = day 0 = night The value of this flag MUST be consistent with dashboard day/night mode HU MUST provide a means for user to manually

⁸ To minimize driver distraction, Google recommends lockout of text input when vehicle is not parked.

⁹ To minimize driver distraction, Google recommends vehicle is parked for FRX.

			choose day or night mode. When in auto mode, value SHOULD be based on ambient light sensor input and care SHOULD be taken to de-bounce the signal to avoid flickering.
<i>Door/Trunk/Hood Status</i>	OPTIONAL	on change	Indicates if open.
<i>Headlights</i>	OPTIONAL	on change	Indicates headlight status (off, on, high beams).
<i>Tire Pressure</i>	OPTIONAL	on change	

Other

Vehicle Data Type	Required	Min Rate	Details
<i>Outside Temperature</i>	OPTIONAL	0.1 Hz	Celsius
<i>Environmental Pressure</i>	OPTIONAL	0.1 Hz	kPA
<i>HVAC - Target /Current Temperature</i>	OPTIONAL	0.1 Hz	Celsius

NOTE

Methods `reportDiagnosticsData` and `reportDeadReckoningData` in the receiver library are deprecated and SHOULD NOT be used in new integrations.

Integrating Navigation

Most mobile devices (MDs) and vehicle head units (HUs) provide navigation services. Because the MD can provide step-by-step navigation and the HU can provide a native (on-board) navigation, the MD and HU **MUST** negotiate navigation focus so the user does not receive concurrent navigation and traffic instructions.

AAP includes two Navigation Focus (NF) modes: **Phone** (navigation provided from MD) and **Car** (navigation provided from HU). On AAP connection the default NF mode is **Car**. AAP enables users to switch navigation focus between the phone and car. However, the **HU always controls navigation focus** and determines which navigation service runs.

The HU **MUST** implement Navigation Focus even if it does not include native navigation.

Navigation Focus Request

A navigation focus request is a message sent from the MD to the HU. The MD makes a request for navigation focus whenever the user changes the navigation state in the MD:

- When navigation on the mobile device is started or restarted, the mobile device makes a navigation focus request to the head unit, requesting NF mode to be phone.
- When navigation on the mobile device is stopped, the mobile device **MAY** make a navigation focus request to the head unit, requesting NF mode to be returned to car.

The HU **MUST** eventually respond to every request with Navigation Focus Notification.

Navigation Focus Notification

A Navigation Focus Notification is a message sent from the HU to the MD that specifies whether the NF Mode is Phone or Car and is used to grant Navigation Focus. The HU:

- **MAY** send a navigation focus notification at any time (for example, when the native navigation service starts or stops).
- **MUST** send a navigation focus notification to the MD in response to a navigation focus request from the device (even if there is no change in navigation focus).

No confirmations from the MD are expected for any of these notifications. The MD will not start phone navigation until it receives a NF Notification with NF mode set to phone. The HU **MUST NOT** run native navigation if the last NF Notification sent was NF mode set to phone.

Policy

The HU SHOULD generally grant the NF Mode that was requested by the MD in a NF Request (that is, if focus is requested for the vehicle, it SHOULD be given to the vehicle; if it is requested for the phone, it SHOULD be given to the phone).

Some special cases worth considering:

- If the HU is actively providing native navigation and the MD requests NF, the HU may choose to stop native navigation and grant NF to the MD, or it may provide a native UI to let the user decide. The RECOMMENDED approach is to stop native navigation and let the MD to take over.
- If a HU is capable of providing native navigation, but is not actively navigating, then it SHOULD immediately grant NF Mode to the phone when requested.
- If the user initiates native navigation while the NF Mode is set to phone then the HU may either issue a NF Notification with NF Mode set to car (and the phone will stop navigation), or it may provide a UI to let the user confirm that phone navigation will end and then send the NF Notification. The HU MUST always use a NF Notification to set NF mode to car before starting native navigation.

Integrating Application Status & Data

The primary visual UI for AAP renders on the mobile device (MD) and projects to the in-vehicle head unit (HU) display as a projection video stream (see [Integrating Video](#)). To enable tighter integration with native purpose-built UI such as instrument clusters and heads-up displays, AAP also provides limited access to app data.

Navigation Data

For vehicles that support next turn information in the instrument cluster, the HU can subscribe to next turn updates from the MD navigation engine.

Navigation Subscription

An HU sends navigation subscription requests to the MD.

Subscription Field	Description
<i>Minimum Interval</i>	Minimum interval (maximum rate) between Next Turn Distance events (in ms).
<i>Next Turn Format</i>	• Image (RECOMMENDED). Instrument clusters that can dynamically load images sent by the MD SHOULD select image for access to high-fidelity next turn iconography from the MD. • Enum. Instrument clusters that cannot dynamically load images SHOULD select enum. The HU MUST select and render a native next turn icon.
<i>Image Options</i>	Required only for HUs that support next turn images. At this time, only square images are supported; includes color depth and resolution.

Navigation Status Event

MDs send navigation status event messages to the HU.

Event Field	Description
<i>Navigation Status</i>	Current status of mobile-based navigation (ACTIVE, INACTIVE, UNAVAILABLE).

Next Turn Event

MDs send next turn event messages to the HU.

Event Field	Description
<i>Next Turn Enum</i>	Enumeration describing the type of turn.
<i>Next Turn Side</i>	Enumeration describing the side of turn (left, right, unspecified)
<i>Next Turn Image</i>	PNG formatted turn image.
<i>Next Turn Road Name</i>	Name of next road (UTF-8).

Next Turn Distance Event

MDs send next turn distance event messages to the HU. These events are sent continuously from MD to HU while navigating, and not necessarily at predictable distance milestones.

Event Field	Description
<i>Next Turn Distance</i>	Distance (meters) to the next turn.
<i>Next Turn Seconds</i>	Estimated time (seconds) to the next turn.
<i>Next Turn Rounded Distance</i>	Distance to the next turn, rounded and converted to the units used when this distance is displayed by the MD navigation application. This value is scaled by 1000.
<i>Rounded Distance Units</i>	The units used on the <i>Next Turn Rounded Distance</i> .

The MD sends distance values in meters and may not time distance events to occur at round number intervals (2km, 1km, 500m, 100m, etc.). The OS adaptation layer (or HU) SHOULD use Next Turn Rounded Distance to display a distance value and units consistent with the MD navigation application.

Next Turn Enum

MDs send next turn enum messages to the HU; the next turn image may have more variations than expressed by the next turn enum value. For example, variations in the sharpness of ON_RAMP turns may be expressed with different images but not be distinguished by this enum.

Enum Value	Has Side	Has Count	Description
DEPART	N	N	Start driving. Used before accurate location or orientation is known.
NAME_CHANGE	N	N	Indicate a street name change without turn.
SLIGHT_TURN	Y	N	Make a 10-45 degree turn.

TURN	Y	N	Make a 45-135 degree turn.
SHARP_TURN	Y	N	Make a 135-175 degree turn.
U_TURN	Y	N	Make a 180 degree turn onto same road.
ON_RAMP	Y	N	Take an on-ramp onto a major road.
OFF_RAMP	Y	N	Take an off-ramp from a major road.
FORK	Y	N	Take a fork while on a ramp.
MERGE	Y	N	Merge with another road.
ROUNDAABOUT_ENTER	Y	N	Enter a large roundabout.
ROUNDAABOUT_EXIT	Y	N	Exit a large roundabout.
ROUNDAABOUT_ENTER_AND_EXIT	Y	Y/N(*)	Enter a roundabout then take the nth exit (count field may potentially be unset).
STRAIGHT	N	N	Indicates a potentially confusing intersection where user goes straight.
FERRY_BOAT	Y	N	Board a boat ferry.
FERRY_TRAIN	Y	N	Board a train ferry.
DESTINATION	Y	N	Indicated arrival at destination.

Navigation Data Example

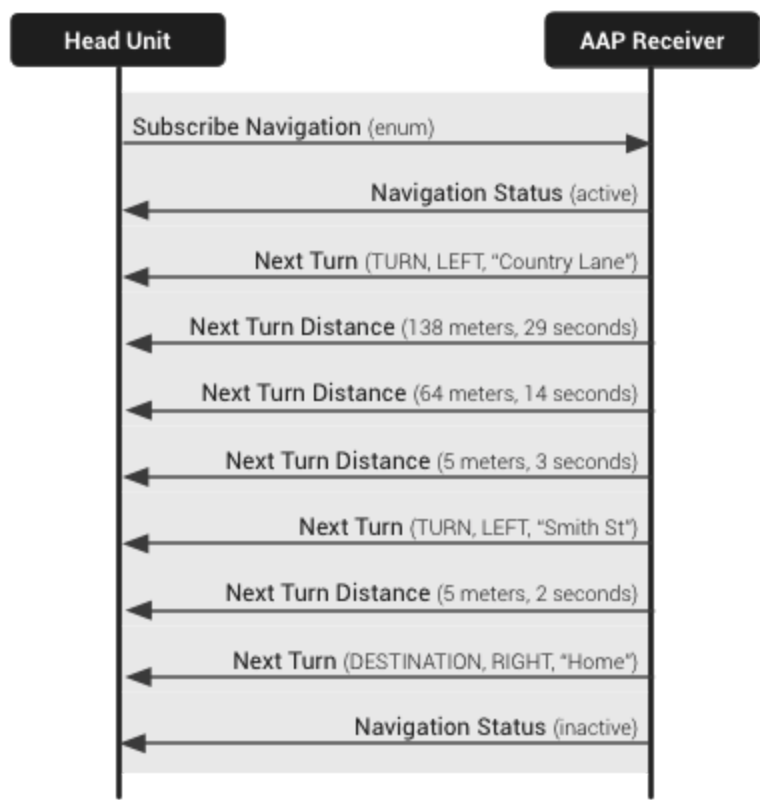


Figure 37. MD navigation.

Accessing Media Data

If the vehicle includes native UIs that display media playback status and/or enable browsing structured media data (such as in an instrument cluster display), the vehicle can access structured media data for media applications on the MD.

Media Playback Status

If the HU subscribes to media playback status, the MD continually sends status notifications whenever media is playing. Notifications can include the following data:

Field	Description
<i>State</i>	PLAYING, PAUSED, STOPPED
<i>Source</i>	The name of the audio (application) source. For example, "Google Play Music", "Pandora", "TuneIn Radio", etc.
<i>Song</i>	Name of the current song/track.
<i>Artist</i>	Name of the current artist.

<i>Album</i>	Name of the current album, if available.
<i>Album Art</i>	PNG image of the album art for the currently playing track, if available.
<i>Playlist</i>	Name of the current playlist, if available.
<i>Duration</i>	Duration of this track in seconds.
<i>Playback</i>	Current progress in track in seconds.
<i>Rating</i>	Rating on a 0-5 scale, if available.
<i>Shuffle</i>	TRUE if shuffle is selected.
<i>Repeat</i>	TRUE if repeat is selected.
<i>Repeat One</i>	TRUE if repeat-one-track is selected.

For details, refer to the `MediaPlayerStatusEndpoint` interface in the [AAP receiver library documentation](#).

Integrating Vendor Extension Channels

Android Auto Protocol (AAP) supports protocol extensions in vendor-specific channels that use AAP as a transport. Vendor Extension Channels (VEC) are completely opaque to AAP, which sends data over the wire without interpreting bits. This section provides an overview of Vendor Extension implementation and includes implementation examples.

Integrators (OEMs, partners) can write proprietary services that run over AAP alongside regular AAP service endpoints, enabling those services to enjoy the benefits that AAP provides. Integrators can add multiple AAP service endpoints, however, to prevent introduction of malware, they **MUST** add authentication and encryption to verify the applications running in the mobile device. (AAP encrypts the data using SSL encrypted to prevent eavesdropping, but integrators can add more authentication as necessary.) For details, refer to the VendorExtension interface in the AAP receiver library documentation.

When to Use VEC

The VEC is designed to address custom, vendor-specific needs—not standard features such as HVAC or FM radio common to all modern vehicles. For standard features, we strongly encourage integrators to work with their Google technical contact and the Google Android Auto team to design and develop applications that run over normal AAP channels.

For non-standard features, use VEC to enable proprietary services to run over AAP. Examples:

- Tunnelling a legacy protocol through the VEC.
- A background service that runs on the mobile device (MD) and collects data from the vehicle even when Android Auto is not projecting to the head unit (HU), then uploads that data to cloud.

Architecture

The Vendor Extension Channel (VEC) architecture requires that a secure connection over SSL is established for all Android Auto protocol traffic, including vendor extension data. The HU specifies the package name of an Android service or app that is allowed to connect to the vendor channel during the Service Discovery phase.

On the MD, when an OEM app is started and requests a VEC connection, Android Auto checks that the service trying to connect to the VEC passes the standard Android Auto permissions checks and has a package name that matches the one declared by the HU vendor extension implementation.

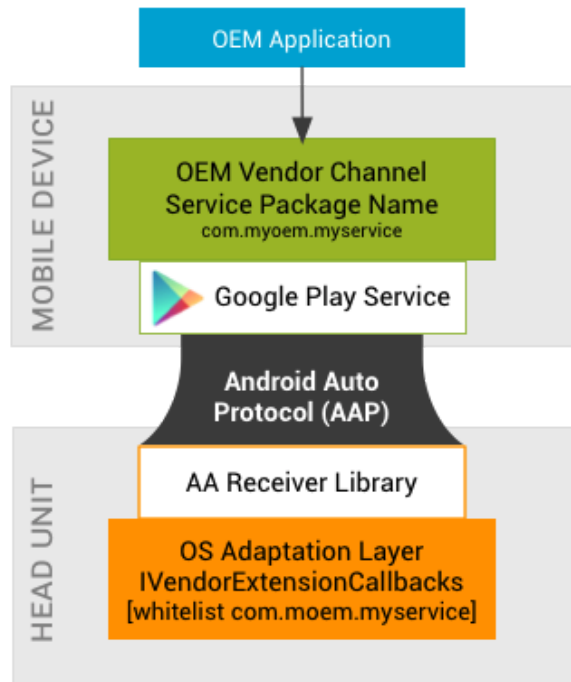


Figure 38. VEC architecture.

If these criteria are satisfied, the nominated OEM service can begin sending and receiving data over the VEC.

NOTE

Standard Android Auto permission checks are handled by the Google Play Store admission procedure (not blacklisted, etc.)

Android Mobile Device

The MD is responsible for initiating the vendor channel connection (i.e. the device is master). An OEM application on the device requests the vendor extension via an OEM-implemented service. Multiple VECs can be active at the same time. Because each VEC is opaque to Google, it is up to the OEM application to multiplex usage of each individual channel among various clients. We recommend:

- Only one (1) Android service interacts directly with any given channel.
- The Android service provides an interface that enables multiple apps to use the VEC.

The Google Play Store manages access to Android Auto APIs by mobile applications on consumer devices. For testing and development purposes, OEMs can ask their Google technical contact to whitelist devices.

Using Verification Certificates

The RECOMMENDED approach of using a single Android service to manage applications' access to the VEC requires the service to verify the signing signature of each app that requests VEC access. VEC access SHOULD be granted only to apps that are signed with the OEM certificate, thus preventing access by unauthorized applications.

To limit VEC connections to your own applications, define a `<permission>` with `android:protectionLevel=signature` and protect the VEC service/provider with that permission.

The OEM Service can use `PackageManager.getPackageInfo()` to access a hash of an APK signing signature, then compare this with a known hash of the OEM certificate held within the service.

Sample Applications

The Android Auto repository includes basic vendor extension examples that echo input. On the **MD**, refer to `//vendor/auto/kitchen_sink/echo-vendor`. On the **HU**, refer to:

- `android-protocol-lib/jni/com_google_android_projection_protocol_VendorExtensionBase.cpp`
- `android-protocol-lib/src/com/google/android/projection/protocol/VendorExtensionBase.java`
- `android-protocol-lib/src/com/google/android/projection/protocol/EchoVendorExtension.java`

Head Unit

On the head unit, vendor channels MUST subclass `VendorExtension` and `IVendorExtensionCallbacks`. When subclassing `VendorExtension`, implement the pure virtual function `VendorExtension::addDiscoveryInfo()`.

Sample Implementation

This section provides pseudocode for the above implementation. It defines the vendor extension in native code; `EchoVendorExtension.java` does the same thing in Java.

First, subclass `VendorExtension` into your own vendor extension class (where `com.vendorname.servicename` is the name used when accessing the VEC):

```

class SampleVendorExtension : public VendorExtension {
    void SampleVendorExtension::addDiscoveryInfo(ServiceDiscoveryResponse* sdr)
    {
        Service* service = sdr->add_services();
        service->set_id(id());
        VendorExtensionService* ves =
        service->mutable_vendor_extension_service();
        ves->set_service_name("com.vendorname.servicename");
        ves->add_package_white_list("com.oem.application.one");
        ves->add_package_white_list("com.oem.application.two");
    }
}

```

Next, subclass `IVendorExtensionCallbacks` to respond to data received from the MD.

```

class SampleVendorExtensionCallbacks : public IVendorExtensionCallbacks {
    // This method gets called when the mobile device sends the HU data.
    int dataAvailable(uint8_t* data, size_t len) {
        // Do something when the data is available. For example send it back
        using sendData.
        mVendorExtension->sendData(data, len);
    }
}

```

Finally, register your vendor extension like any other service during the initialization sequence.

```

galReceiver->init();
... Initialization code ...
VendorExtension* endpoint = new SampleVendorExtension(serviceId,
galReceiver->messageRouter());

SampleVendorExtensionCallbacks* callbacks = new SampleVendorExtension(endpoint);
endpoint->registerCallbacks(callbacks);
galReceiver->registerService(endpoint);
... Other Initialization code ...
galReceiver->start();

```

On the MD, a single Android service SHOULD handle the VEC data processing, irrespective of the number of OEM apps that may ultimately consume the data. The HU vendor MUST determine the package name of this service in advance and set in the `VendorExtension::addDiscoveryInfo()`, which corresponds to the `add_package_white_list()` call in the example. The `dataAvailable()` callback is executed when data is received from the MD via the VEC. To respond, the HU SHOULD call `sendData()`.

Use Cases

The following use cases assume the user has installed the OEM app on the MD and granted permission to access car data through the Google Play Store Install App permissions screen.

Autostart Application

When an MD connects to an HU that supports Android Auto, the OEM can automatically start a telematics or similar app. This app does not have a usable interface on the device because of the Android Auto lock screen, but the app could, for example, launch an app-based service. The app could then perform a background action such as collect data from the HU over the VEC and send it off to a cloud store for the user to view later.

Autostart includes two steps:

1. On the HU, determine whether the MD is Android Auto compatible.
2. On the MD, initialize the VEC and auto-start the correct app.

Determine Android Auto Compatibility

1. HU sends the Android Auto AOA string.
2. If the MD responds in AOA Accessory mode, it's an Android device.
3. HU begins Android Auto connection.
4. Does the MD send a ByeBye message over the Android Auto protocol?
 - If NO, the device supports Android Auto. Begin [initializing the VEC](#).
 - If YES, it's common to re-establish an AOA/USB connection to the MD for another purpose, such as connecting the HU directly to a telemetry app already on the MD.
 - i. MD sends ByeBye message over Android Auto protocol. Because the AA protocol is implemented in Google Play Services, the ByeBye message is sent even from devices running older versions of Android.
 - ii. Reset USB on the HU. This allows the HU and MD to reconnect without Android Auto AOA mode.
 - iii. HU sends alternate AOA string to connect directly to the app on the MD.

Initialize Vendor Extension Channel

After validating the MD as Android Auto compatible, you can initialize the VEC. Vendor Extension Channels MUST be pre-built into the HU, so initialization tasks occur exclusively on the MD. The communication is between the Android Auto CarService component and the

OEM App. Before starting, ensure:

- The OEM application package name is in the package whitelist of a VEC on the HU. The whitelist ensures that OEM X application will be allowed to communicate over the VEC for the OEM X HU, but not with OEM Y HU.
- OEM application listens for the `VENDOR_CHANNEL_READY` broadcast and declares the permission `Car.PERMISSION_VENDOR_EXTENSION` in its manifest.

During initialization, the following events occur:

1. CarService sends a broadcast announcing `VENDOR_CHANNEL_READY` to each package on the channel's whitelist¹⁰.
 - Any app that receives the Directed Broadcast will be **automatically started** by Android, even if the app is not yet running.
 - If an app that is not whitelisted to open the VEC attempts to open the VEC, the `CarVendorExtensionManager` throws a `SecurityException`.
2. OEM App responds by establishing a client connection to Car API in Google Play Services: `mCVEM = Car.CarApi.getCarVendorExtensionManager(mCarApiClient, SERVICE_NAME);`
3. OEM App registers a listener on the `CarVendorExtensionManager` which calls `onData()` in the listener when data arrives.
4. HU and OEM App begin sharing data over the VEC. The `CarVendorExtensionManager` has a `sendData()` call for sending data.

Frequently Asked Questions

Can a specific app on the MD be started automatically when connected to the HU?

Yes. See [Autostart Application](#).

Can an OEM app on the MD display on the phone screen while Android Auto is active?

No. An active Android Auto connection avoids distracting situations by displaying a lock screen on the MD. Instead, the application SHOULD use the OEM SDK to display a native Android Auto experience on the head unit display or run in a service mode to gather information that drivers can look at after they disconnect their MD from the HU.

Can one Vendor Extension Channel support more than one app?

AAP does not handle multiplexing over the Vendor Extension Channel. For this reason,

¹⁰As of Google Play Services version 7.8, the VEC is not identified by name to the package.

Google recommends that OEMs create a service app on the device that handles all communications with a given VEC, and that this service app handles one-to-many communications between the VEC and other apps on the phone. Note also that multiple VECs can exist in parallel.

How does an OEM-specific app get installed when a specific VEC is detected on the HU?

Auto-detecting the correct app to install from the Google Play Store is not yet supported. Users of the OEM app MUST install the app themselves. Google plans to add this capability in a future release of Google Play Services. For users, the likely flow will be to ask them during the first-run experience whether they'd like to install the appropriate OEM app.

How should the OEM-specific app react when the VEC is closed? Is the response different for typical cases (such as an unplugged USB cable) vs. atypical cases (such as the Android Auto app on the phone crashes)?

Retry and disconnect logic SHOULD be part of any app that communicates with a data service like the VEC. An OEM SHOULD decide how to deal with each case; the Android Auto Protocol simply provides an opaque transport for data. For example, if AAP is not available because of a failure in the Head Unit OS Abstraction Layer, an OEM telemetry app may switch to a different communication method, e.g., data over Bluetooth, legacy protocol over USB, different AOA ID over USB.

Integrating Radio

Radio is a new, OPTIONAL API added in AAP v1.2 that enables control the vehicle radio from within AAP. Users do not need to exit the Android Auto experience to access radio playback control via the native head unit (HU) user interface. In a high level view of the radio control flow from AAP to the HU Radio, the HU plays radio audio directly (audio stream does not get routed to the MD). To enable Radio API for AAP, integrators SHOULD implement `RadioEndpoint` and `RadioCallbacks` in the Receiver Library.

Radio Service Discovery

During service discovery (performed on each MD connection), HUs that support the Radio API MUST include radio service as part of the discovery response. This process tells AAP that the radio API can be used. The service discovery response contains a message that describes the HU's radio properties. The properties advertised by the HU are used by AAP to determine available types (AM/FM, HD, etc) and the features to enable in the radio app.

Radio Properties

The HU notifies the MD of the Radio properties the HU supports using a `RadioProperties` instance, defined for each radio type supported by the HU.

RadioProperties Field	Description
<i>RadioType</i>	Type of Radio: AM, FM, AM_HD, FM_HD, DAB or XM (Only AM and FM are currently supported).
<i>Range</i>	Min and Max supported frequency values.
<i>channel_spacings</i>	All supported channel spacing configurations (resolution of a frequency step).
<i>channel_spacing</i>	Current channel spacing.
<i>background_tuner</i>	Indicates if a background tuner is available (for background scans).
<i>RadioRegion</i>	Radio region.
<i>RdsType</i>	RDS (Radio Data System) Type. Also known as Radio Broadcast Data System (RBDS) in the US.
<i>af</i>	Alternative frequency switching (RDS) support.
<i>ta</i>	Traffic announcements (RDS) support.
<i>TrafficService</i>	Indicates if traffic service available.
<i>mute_capability</i>	Indicates if "radio only" muting is supported.
<i>radio_state</i>	The state of the radio at the time when MD connected to HU.

Messaging Between HU and MD

Messages passed between the HU and MD consist of requests from the MD, corresponding responses from the HU, and notifications from the HU when state changes. For an overview of messages, see below; for specific parameter values, refer to the Receiver Library source.

Function	Message	Details
<i>Radio Source</i>	• EnableRadioSourceRequest • EnableRadioSourceResponse	Enables radio sources.
<i>Active Radio</i>	• SelectActiveRadioRequest • ActiveRadioNotification	Changes active radio (i.e. from FM to AM). HU SHOULD tune to the previously set station for the specified radio.
<i>Step Channel</i>	• StepChannelRequest • StepChannelResponse	Steps radio channel up and down. Step resolution is determined by the channel spacing of the band. [†]
<i>Seek</i>	• SeekStationRequest • SeekStationResponse	Seeks next station (up or down). [†]
<i>Scan</i>	• ScanStationsRequest • ScanStationsResponse	Scans available stations while previewing each station for an interval of time, until user stops scan. [†]
<i>Tune to Station</i>	• TuneToStationRequest • TuneToStationResponse	Tunes to specified band/channel/HD station (if supported). [†]
<i>Station Info Notification</i>	• RadioStationInfoNotification	HU sends this to the MD when station info or meta data changes (i.e. tuned to new station, or RDS data changed).
<i>Get Program List</i>	• GetProgramListRequest • GetProgramListResponse	Returns a list of all available stations. Since this operation can be time consuming, the HU may stream results back to the MD by sending multiple response messages. If a background tuner is available, this process uses that tuner; if unavailable, playback may be interrupted and radio

		app may mute radio to prevent sound from being played during program list retrieval.
<i>Cancel</i>	• CancelRadioOperations Request • CancelRadioOperations Response	Cancels all outstanding radio operations.
<i>Configure Channel Spacing</i>	• ConfigureChannelSpacing Request • ConfigureChannelSpacing Response	Used in regions where channel spacing is user configurable.
<i>Station Presets</i>	• StationPresetsNotification	Sent when station presets have been changed by the user.
<i>Mute</i>	• MuteRadioRequest • MuteRadioResponse	Mutes radio
<i>Traffic Update</i>	• GetTrafficUpdateRequest • GetTrafficUpdateResponse	Returns traffic updates.

† Tuning to the next channel is performed asynchronously; a separate RadioStationInfoNotification is sent to the MD each time the radio tunes to the new station.

Example Radio Sequences

The following examples illustrate common radio use cases.

Radio Playing During MD Connection

In this example, the MD connects while radio is playing on the HU. The HU takes audio focus and notifies MD of the active radio station.

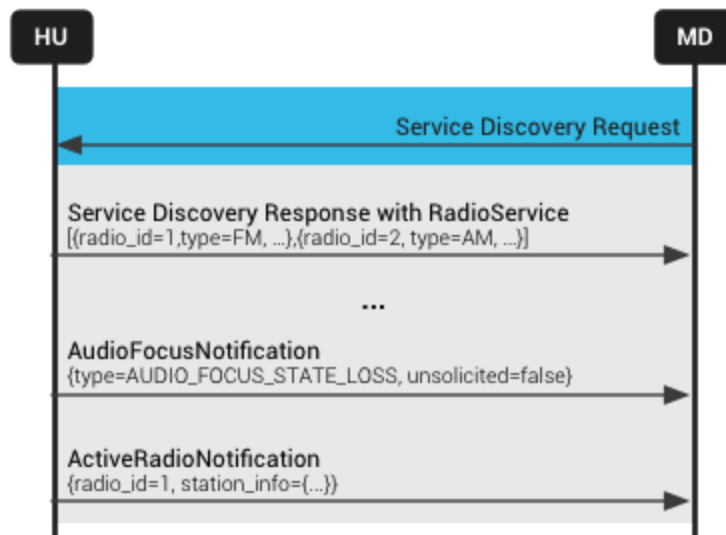


Figure 39. Radio service discovery sequence.

Tuning to Station in Projection Mode

In this example, a user sets the radio to 95.7 FM in projection mode.

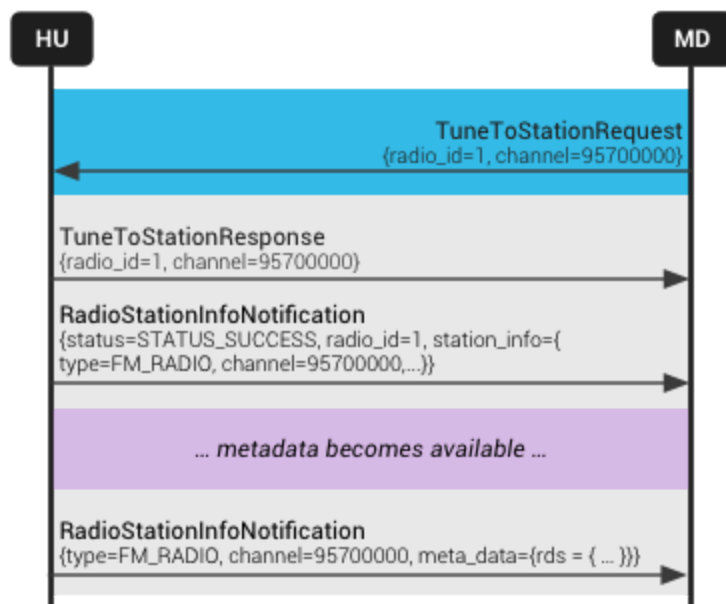


Figure 40. Tuning to station in projection mode sequence.

Tuning to Station in Native Mode

In this example, a user selects station preset 1 to in native mode.

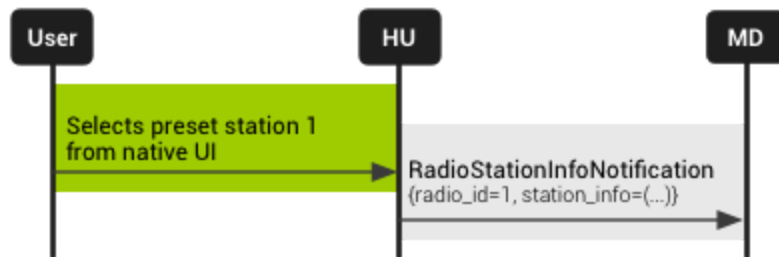


Figure 41. Tuning to station in native mode sequence.

Scanning Stations in Projection Mode

In this example, a user scans stations in projection mode.

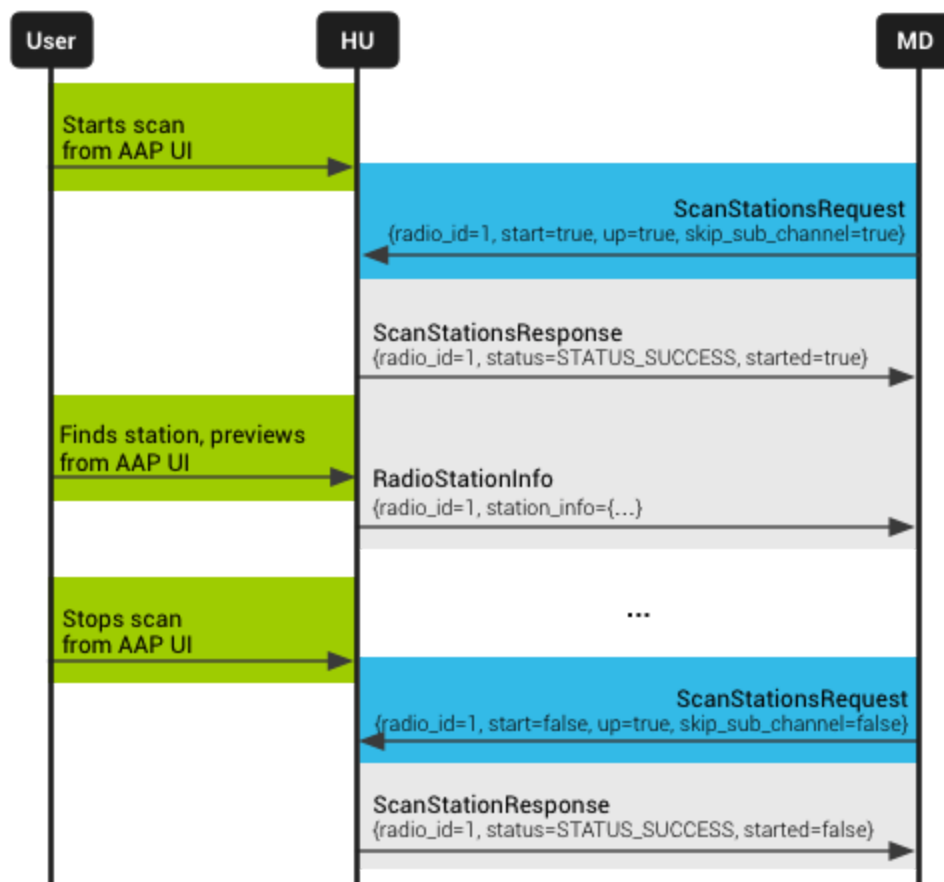


Figure 42. Scanning stations in projection mode sequence.

Audio Focus When Integrating Radio

Audio focus for AAP is handled in the same manner as for other sources. If radio source is enabled, the MD MUST release audio focus to enable radio playback. The following diagram illustrates how audio focus is handled when playing radio.



Figure 43. Radio Audio focus.

Radio Hard Buttons

The HU SHALL handle events for all radio controls:

- scan up/down
- seek up/down
- step up/down
- preset next/prev
- preset #
- radio mode (i.e. AM/FM, etc)

The HU notifies the MD via corresponding notification messages (i.e. StepChannelResponse, SeekStationResponse, ScanStationsResponse, RadioStationInfoNotification etc).

Appendix A: Vehicle ID

A valid Vehicle ID SHOULD exhibit the following properties.

Unique ID per HU

Android Auto uses Vehicle ID to determine if the First Run Experience has run on on a given head unit (HU). The Vehicle ID MUST be unique to the individual HU, ie. not shared across multiple HUs of the same make/model. The value can be random, as long as the same value is retained within a HU until factory reset occurs.

Minimum 64 Bits

To reduce the chances of ID collision, Vehicle IDs SHOULD have at least 64 bits. That is, at least 2^{64} values are possible within the definition of the ID.

For example, if an ID can be made up of the following characters: [A-Z, a-z, 0-9], that gives 62 possible values for each character, which can be represented by 6 bits ($2^6=64$). To exceed 64 bits, at least **11 characters** would be needed (roundup $64/6$).

Likewise, if an ID consists of [A-Z, 0-9], each character has 36 possible values, which is roughly 5 bits ($2^5=32$). The ID would need **13 characters** to cover 64-bit space (roundup $64/5$).

Example: Generate Vehicle ID

This example generates a valid Vehicle ID without a factory-written serial number. For a random number generated at first boot (or factory reset), if the HU has writable storage (or non-volatile space):

1. When the sink implementation starts up, it looks for an identity file (such as `/var/gal/identity`).
2. If the file doesn't exist, read (or generate) random 128 bits (such as `/dev/urandom`) and write it to an identify file.
3. Use ID as the HU identity going forward, until the file is deleted (such as during a factory reset).

Appendix B: Document History

Revision	Date	Changes
1.0.0	5 Sep 2014	- Applying new template. - Name change (GAL to AAP). - Adding Audio Focus details. - Remove "Use Cases" section. - Remove Wi-Fi transport details. - Remove Radio. - Rationalize with Protocol. - Added Authorize Connection sub-section on Service Discovery. - Additional detail on navigation sensor requirements. - Rewrote connection establishment section and Bluetooth related sections.
1.0.1	29 Sep 2014	- Clarified required resolutions: MD may send 800x480 at any time. - Changed name of product to Android Auto. - Clarified A2DP handling. - Clarified navigation sensor exception process.
1.0.2	08 Jul 2015	- HFP connection not required to complete AAP handshake. - Added 500ms timeout for MD waiting for Audio Focus grant from HU. - In Configuring AOAP table, changed manufacturer from Google to Android and Description from <empty> to Android Auto. - Updated Current Gear information to specify "Park" criteria. - Removed 1 Hz GPS requirement. - Clarified that vehicle data MUST never be faked. - Added description to Vehicle_ID. - Clarified call handling and HFP/button send requirements. - Changed Current Gear from required to OPTIONAL, clarified usage. - Clarified conflict resolution behaviour when Android Auto connected with navigation running on both MD and HU. - Separate video resolution from frame rate. - Clarified desired semantics of day/night mode flag. - Clarified AAP session persists even if not visible on HU screen. - Updated audio stream requirements: PCM required, AAC-LC OPTIONAL. - Clarified KEYCODE_SEARCH SHOULD be reported if a physical speech button exists. - Touch events allowed while native OSD visible in some cases. - Clarified navigation focus handling. - Added table to clarify channel terminology to stream terminology. - Clarified position on RSAP, PAN, MAP Bluetooth profiles. - ASR (formerly VR) section split into updated reqs and recommendations. - ASR triggering (HU vs MD) policy updated. - Added open source components.

		- Added guidance on native HMI during First Run prior to MD video projection. - Changed RECOMMENDED audio encoding from AAC to PCM.
1.2.0	03 Dec 15	- A2DP: clearer statement that it can be enabled during AAP connection, and provide an example of audio focus handling. - h.264 decoder SHALL handle variable rate stream - Compass sensor may include gyro stabilization - Clarified the policy on requirements, frequencies and axes for a set of sensors: light OPTIONAL, explicitly state that 1-axis accel and gyros are ok, relax frequency requirements to 1 Hz. - Clarified on gyro, accelerometer sensors. - Added GPS location clarification. - Clarified differences between next turn image and next turn enum - Clarified that voice button SHOULD be a dedicated button. - Added more precise minimum screen size requirements - Added suggestion that additional devices charge to avoid complicated multiple device scenarios. - Clarified information that SHOULD be sent in Head_Unit_Make, Head_Unit_Model, Head_Unit_Software_Build and Head_Unit_Software_Version. - Clarified that during connection setup, video focus MUST NOT be granted to the MD by the HU (VIDEO_FOCUS_PROJECTED) until VideoSink::setupCallback() has been called. - Added guidance on Audio Focus request handling in cases where HU setup takes time. - Added that MD plugged in before car powered on SHOULD also initiate connection. - Changed total combined latency target to 250ms from 200ms inline with performance test requirements (Figure 15). - Added guidance around audio buffering. - Clarified handling of ASR hard keys during VR sessions. - Added description of 00000 drive level setting. - Clarified KEYCODE_BACK does not cause Android Auto to lose video focus. - Added more detail to description of KEYCODE_ROTARY_CONTROLLER. - Added documentation for Vehicle Data interfaces available from GAL v1.0 but not covered in HUIG. - Updates for AAP v1.2 - New video focus Native Transient. - Session Configuration added to Service Discovery parameters. - HU can report pixel aspect ratio to MD to receive video corrected for non-square pixels. - Temporary and permanent sensor errors can be reported to MD when identified by the HU. - HU SHOULD report transformations performed on GPS data using locationCharacterization.

		○ Added new key events. ○ Added more gear states. ○ Added documentation for integrating touchpad as primary input device. ● New doc template applied. ● Updated BT connection diagrams for previously-paired and never-paired MDs. ● Removed requirement that Hands-Free Profile (HFP) connections to other devices be disconnected in preparation for BT pairing. ● Added new section Integrating Vendor Extension Channels. ● Added new section Integrating Radio. ● Added new diagram on determining if MD supports AAP.
1.2.1	29 Mar 16	● HU MUST notify user if max number of BT HFP connections has been exceeded, thus blocking AAP connection. ● Added USB power, USB hub requirements. ● Clarified AAP Connection Flow requirements to avoid SSL context. failures and timeout for slower phone response. ● Removed references to h.264 Main Profile in favor of Baseline Profile. ● Updated to RFC2119 language. ● Changed audio setup latency from 200ms to 500ms. ● Updated some old diagrams.
1.3.0	10 Aug 16	● Added VR button guidance ● BT HFP recommended during FRX, not mandatory ● Preferred behaviour when max BT connections reached. ● Revised USB connection flow descriptions