

Advanced Driver Information Technology

BDCL – Component Specification

1 Document Information

Document ID	BDCL-001
Document Revision	C
Document Type	DS
Document Owner	K. Gerhards
Document Authors	P. Govindaraju
Document Classification	Confidential
Document Creation Date	18-Oct-2016
Document Release Update	24-Jan-2017
Last Update	07-Dec-2018

3 Review Information

Revision	Reviewed by	Responsible SPL	Approval Date	Signature
A	M. Methner, D. Girnus, J. Harder.			
B	M. Methner, D. Girnus, P. Acar			

Table of Contents

Document Information	1
Review Information.....	1
1 Executive Summary	4
2 Indices and Tables	4
2.1 Change History	4
2.2 Glossary.....	5
2.3 References	5
2.4 Table of Figures.....	5
3 Introduction.....	7
3.1 Use Cases	7
4 Concept	9
4.1 Architectural Overview.....	9
4.2 Logical Overview	10
4.3 Component View	11
4.4 Physical Connection of MD	11
4.4.1 Physical Connection of Android Device.	11
4.4.2 Physical Connection of iOS Device	12
4.4.3 Physical Disconnection of Android Device	13
4.4.4 Physical Disconnection of iOS Device.....	14
5 Design	16
5.1 Transport Handling	16
6 Interfaces for BDCL Platform	17
6.1 Adapter Interfaces Overview	17
6.2 Adit Wrapper Adaptation	17
6.2.1 Baidu CarLife Initialization.	18
6.2.1.1 Baidu CarLife Session Initialization for Android Device.	20
6.2.1.2 Baidu CarLife session Initialization for iOS Device.	21
6.2.1.3 Baidu CarLife session termination for Android Device.	21

37	6.2.1.4	Baidu CarLife session termination for iOS Device.	22
38	6.3	Video Sink Adaptation	23
39	6.3.1	Video Sink Configuration	24
40	6.3.2	Video Adaptation Initialization	25
41	6.4	Audio Sink Adaptation	27
42	6.4.1	Audio Sink Configuration	28
43	6.4.2	Audio Sink Adaptation Initialization	30
44	6.4.3	AIL Audio Sink Processing	31
45	6.5	Audio Source Adaptation	33
46	6.5.1	Audio Source Configuration.....	34
47	6.5.2	Audio Source Adaptation Initialization	35
48	6.5.3	Audio Source Processing	36
49	6.6	Input Source Adaptation	38
50	6.6.1	USPI Input Source	38
51	6.6.2	Input Source Initialization	39
52	6.6.3	Input Source Touch Input Thread	40
53	6.7	Debugging	40
54	6.7.1	DLT	40
55	6.7.2	DLT Context.....	41
56	6.8	Channel ID	42
57	6.9	Deviation in the BDCL app behavior against Baidu specification	42
58			
59			

1 Executive Summary

This document explains the interfaces and internal design of the components for BDCL (Baidu CarLife). The design is based on the Baidu CarLife Integration Specification V1.2.9 [Baidu Spec]. It contains all interface descriptions required to implement custom adapters.

As the integration of the Baidu code into the ADIT platform progresses, this document will be continuously updated.

2 Indices and Tables

2.1 Change History

Revision	Date	Author	Description
A01	18-Oct-2016	P. Govindaraju	Initial release.
A04	27-Oct-2016	M. Methner	Reworked security chapter
A05	16-Nov-2016	P.Govindaraju	161026_RV003_BDCL_Draft_Specification review findings are fixed.
A	29-Nov-2016	P.Govindaraju	A version released to Bosch MC.
B01	9-Jan-2017	M. Methner	Removed reference to EA model
B02	13-Jan-2017	P.Govindaraju	Removed the unreferenced chapter.
B03	23-Jan-2017	P.Govindaraju	Review findings has fixed
B	24-Jan-17	P.Govindaraju	B version is released.
C01	10-Feb-2017	P.Govindaraju	Video and Audio Adaptation chapter added.
C02	13-Feb-2017	P.Govindaraju	Video and Audio adaptation initialization sequence diagram added.
C03	15-Feb-2017	P.Govindaraju , M. Methner	Updated video and init sequence diagram
C04	20-Feb-2017	P.Govindaraju	Updated the media audio initialization sequence diagram
C05	15-Mar-2017	P.Govindaraju	Updated the Init sequence diagram and touch adaptation.
C06	17-Mar-2017	P.Govindaraju	Review Findings are Fixed.
C07	10-Apr-2017	M. Methner	Updated AiAudioSink chapter
C08	13-Apr-2017	P.Govindaraju	Updated Video Adaptation Initilazation
C09	30-Jul-2018	P.Govindaraju	Updated the document from beginning till Audio sink configuration
C10	10-Oct-2018	P.Govindaraju	Updated the document completely added Audio source and touch endpoint details
C11	07-Dec-2018	P.Govindaraju	Fixed the review findings.
C	12-Dec-2018	P.Govindaraju	C version is released

69 **2.2 Glossary**

Term	Description
iAP2	<i>iPod Accessory Protocol version 2.</i>
BDCL	<i>BaiDu CarLife</i>
EAP	<i>External Accessory Protocol</i>
HU	<i>Head Unit</i>
MD	<i>Mobile Device.</i>
MC	<i>Mother company of ADIT – BOSCH and DENSO</i>
SPI	<i>Smartphone Integration</i>
USPI	<i>Unified Smart Phone Integration</i>
AIL	<i>Audio Interface Library</i>
FeatureDiscovery	<i>Component by ADIT which provides the notifications to applications whenever the Mobile Device is connected and disconnected from the USB port.</i>

71 **2.3 References**

Label	Owner	Title	Document No.
[AOAP]	Google	Android Open Accessory Protocol Specification	http://source.android.com/accessories/protocol.html
[ADIT AOAP]	ADIT, T. Fickel, D. Girus	ADIT implementation of the Android Open Accessory Protocol	AOAP-001-DS
[EA Model]	P. Govindaraju	BDCL UML Model	Baidu-01-DS-A10_baidu_UML_Model
[Baidu Spec]	Baidu	Baidu CarLife	Baidu CarLife Integration Specification v1.2.9

73 **2.4 Table of Figures**

74	Figure 1 – BDCL Use Case Overview.....	7
75	Figure 2 – BDCL Use Case Breakdown.....	8
76	Figure 3 – Architecture Overview	9
77	Figure 4 – Logical Overview	10
78	Figure 5 - BDCL Library Component.....	11
79	Figure 6 - Physical connection of Android device	12
80	Figure 7 - Physical connection of iOS device.....	13
81	Figure 8 - Physical disconnection of Android device.....	14
82	Figure 9 - Physically disconnection of iOS device	15

83	Figure 10 – Transport Handling Class Model Overview.....	16
84	Figure 11 – Baidu Platform Adaptation	17
85	Figure 12 - ADIT Wrapper Overview	18
86	Figure 13 - Baidu CarLife session initialization for Android Device.....	20
87	Figure 14 - Baidu CarLife session initialization for iOS device.....	21
88	Figure 15 - Baidu CarLife session termination for Android Device	22
89	Figure 16 - Baidu CarLife session termination for iOS device	23
90	Figure 17 - Video Sink Overview	23
91	Figure 18 - Adit Video Adaptation	24
92	Figure 19 - Video Sink Initialization	26
93	Figure 20 - Audio Sink Overview	27
94	Figure 21- bdcl Audio Sink Adaptation	28
95	Figure 22 - Audio Adaptation Initialization.....	31
96	Figure 23 - Ail Audio Sink processing	32
97	Figure 24 - Audio Source Overview	33
98	Figure 25 - bdcl Source Adaptation.....	34
99	Figure 26 - Audio Source Adaptation Initialization	36
100	Figure 27 - Audio Source Processing.....	37
101	Figure 28 - Input Source Adaptation	38
102	Figure 29 - USPI Input Source Endpoint.....	39
103	Figure 30 - Input Source Initialization.....	39
104	Figure 31 - Touch Input Thread.....	40
105	Figure 32 - iOS device behavior for pause and play Media Audio	42
106	Figure 33 - Android device behavior for pause and play Media Audio	43
107	Figure 34 - iOS device behavior for Media Audio track change	43
108	Figure 35 - Android device behavior for Media Audio track change	44
109		
110		

3 Introduction

3.1 Use Cases

BDCL allows the user to connect an Android/Apple device to the Head Unit and use the functions of the Baidu CarLife App running in the connected mobile device through the head unit's peripherals.

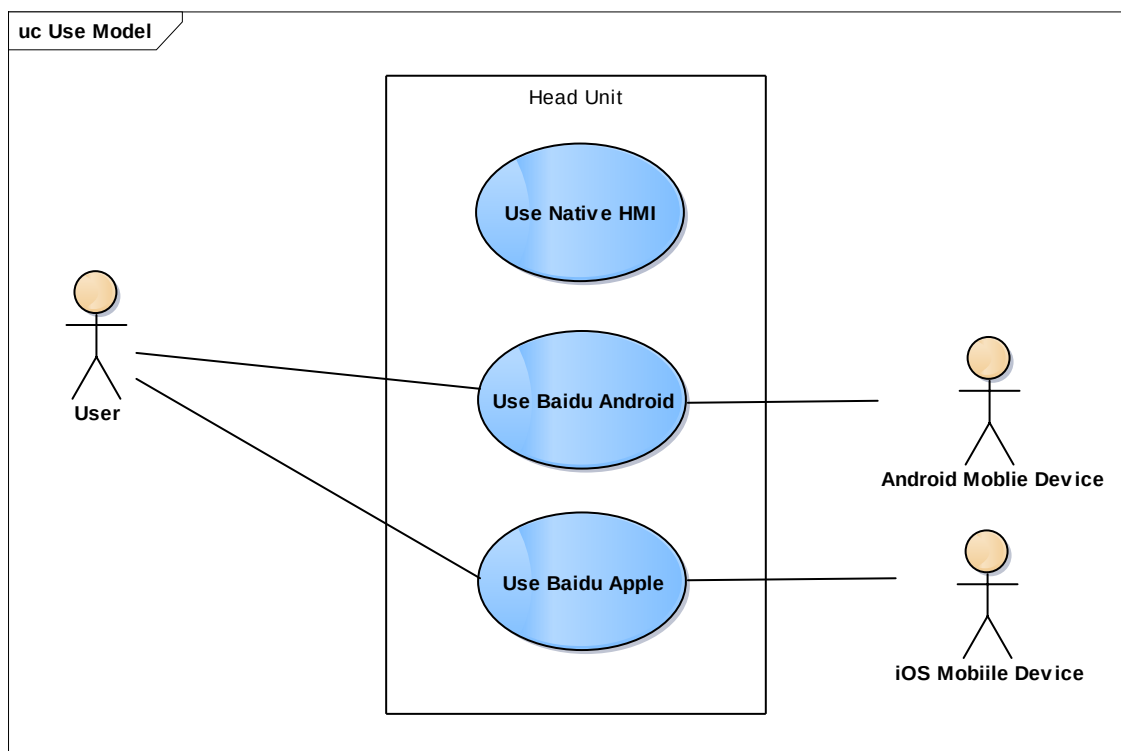


Figure 1 – BDCL Use Case Overview

In Figure 1 three top-level use cases within the Head Unit are considered. External actors are the User and the Android/Apple Mobile Device. When a Mobile Device is connected to the Head Unit the User can use native HMI features and BDCL features. The Mobile Device also triggers some of the BDCL uses – e.g. an incoming phone call.

A breakdown of the use BDCL use case in Figure 2 shows roughly which kind of applications are available and interact with the system.

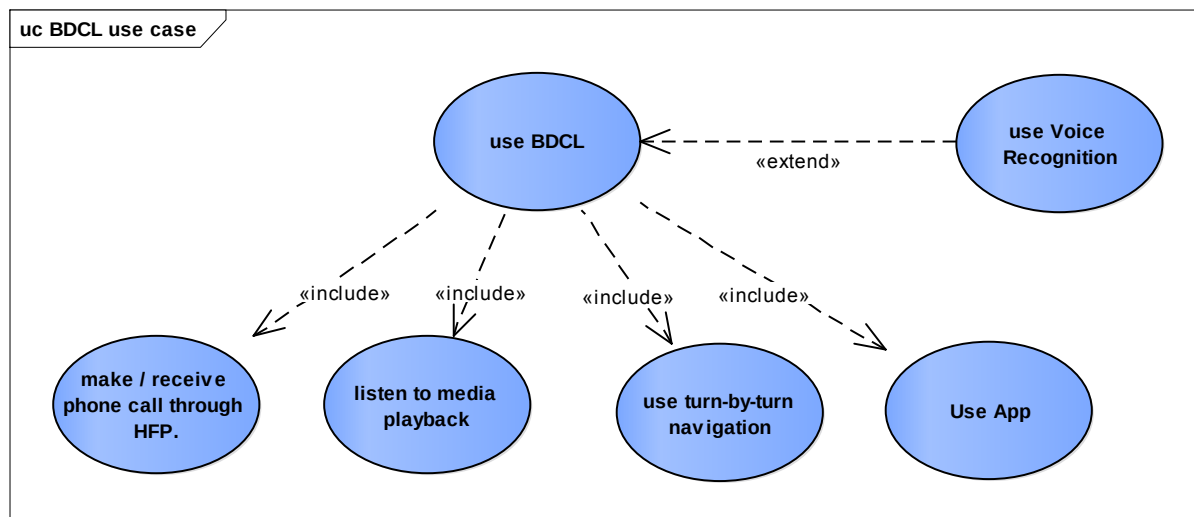


Figure 2 – BDCL Use Case Breakdown

use turn-by-turn navigation: BDCL includes a turn-by-turn navigation application.

listen to music: BDCL includes music playback which can be controlled through BDCL display.

Voice Recognition: Voice data from mic record in Head Unit to MD can be sent from BDCL.

Phone call: We can make / receive phone call through HFP. Phone call audio packets will not be streamed through CarLife Stack.

Use App: Apps can be used through CarLife.

4 Concept

4.1 Architectural Overview

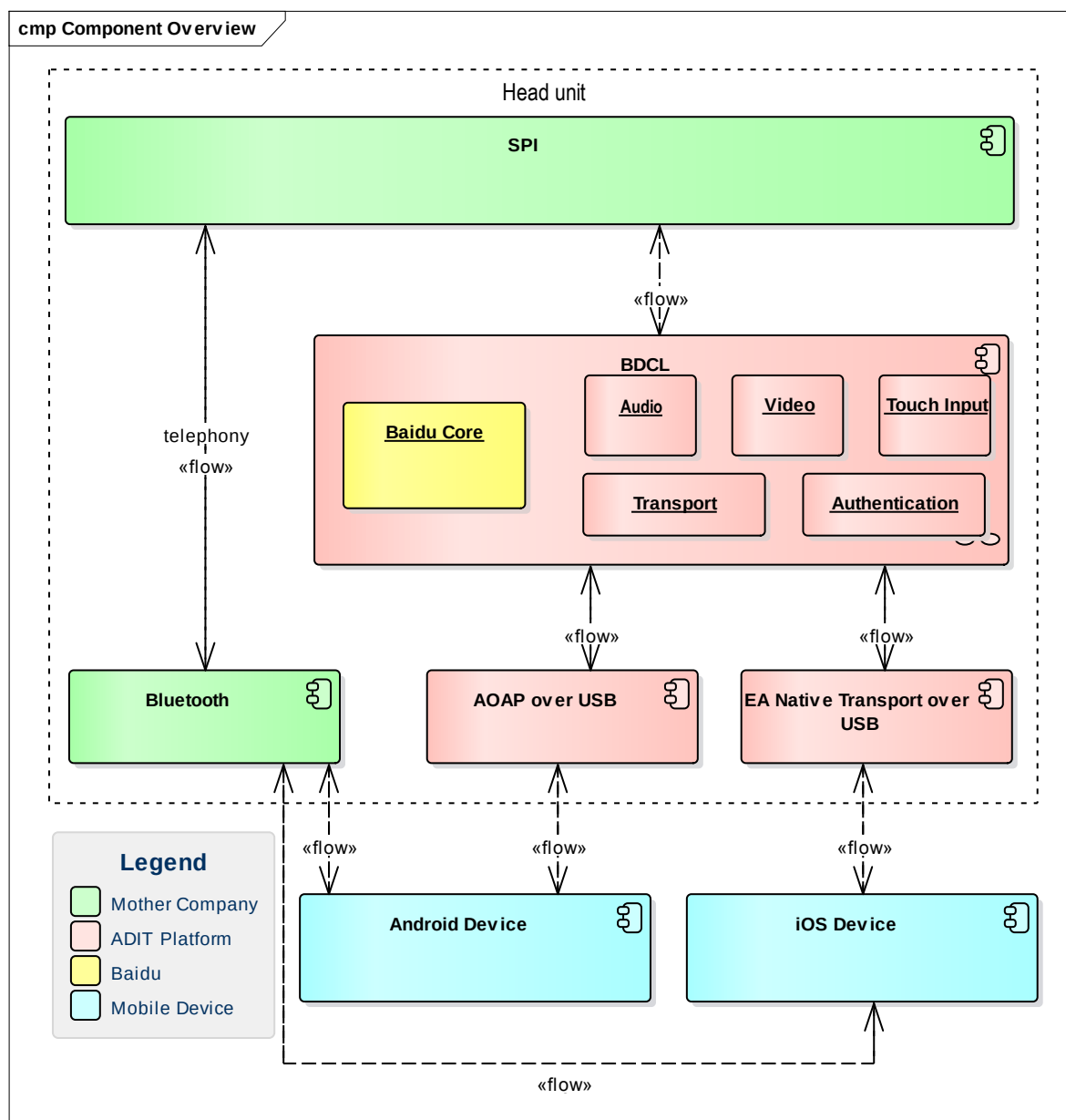
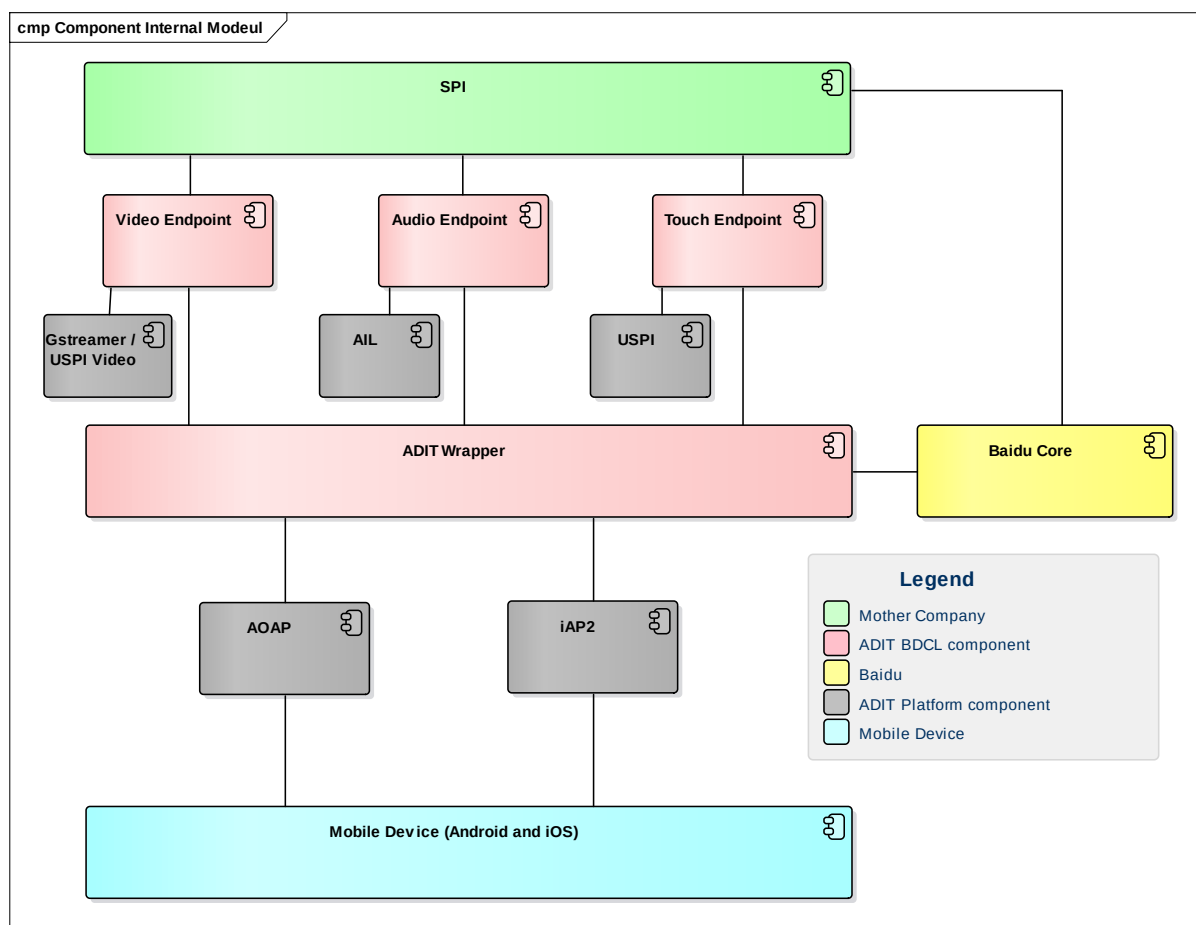


Figure 3 – Architecture Overview

SPI services of MC rely on BDCL component for Transport, Audio, Video and Touch Input. Other services such as Button Input, Bluetooth Telephony are to be developed by the MC themselves.

141 **4.2 Logical Overview**142 **Figure 4 – Logical Overview**

143 The Figure 4 gives an overview of the involved logical elements in the Head Unit for BDCL. The ADIT
 144 BDCL Component library represents the components on the ADIT platform for this feature. It will be
 145 accessible via platform-independent adapter interfaces for audio/video, input events and control
 146 notifications and methods. A Mother Company component (which is termed as MC application in Figure
 147 4 – Logical Overview) in combination with a media framework (e.g. GStreamer) is imagined to handle
 148 those interfaces. The ADIT Wrapper Component will use Baidu Core which basically maintain the
 149 communication session with connected mobile device. The Baidu Core communicates with Mobile
 150 device with two different protocols depending on connected MD. In case of Android device Baidu Core
 151 communicates with AOAP session whereas for iOS device the communication happens through EA
 152 Native Transport.
 153

4.3 Component View

BDCL library contains two parts Baidu Core provided by Baidu itself, which mainly maintains the Baidu Session, which depends on the Google Protobuf to serialize the Baidu messages. Also MC application can directly access the Baidu API's. The second part to be implemented is ADIT Baidu Endpoints, which connects the Core with platform-specific components. The interfaces in Figure 5 – Audio/Video, Input and Control are designed as plug-able adapters so that it is possible to replace parts of the implementation depending on the platform.

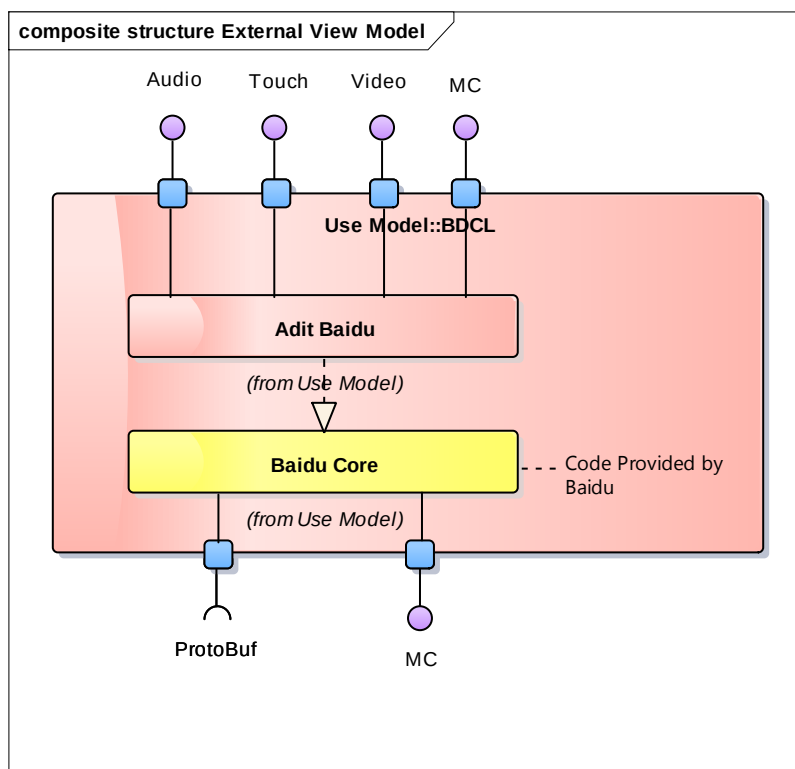


Figure 5 - BDCL Library Component

4.4 Physical Connection of MD

This chapter explains the sequence of operations after connecting the MD to HU and before starting the CarLife session.

4.4.1 Physical Connection of Android Device.

Figure 6 how Android device is detected and switched to AOAP mode using the FeatureDiscovery component. BDCL can be started after the MD was switched to AOAP mode. This can be done by sending some vendor specific commands like #51 to get AOA Version, #52 to get identify info and #53 to switch the device to AOAP mode for Baidu CarLife.

Requirement to start Baidu CarLife is that MC application has to provide Accessory ID and Device ID to aditCoreSurrogate once it successfully receives the switchedCb(). Accessory ID and Device ID will be provided by libaoap once the MD changes its role to accessory mode.

MC may use some other device detection technology in place of FeatureDiscovery to detect the connection and disconnection of the device. It is advisable from ADIT to use FeatureDiscovery.

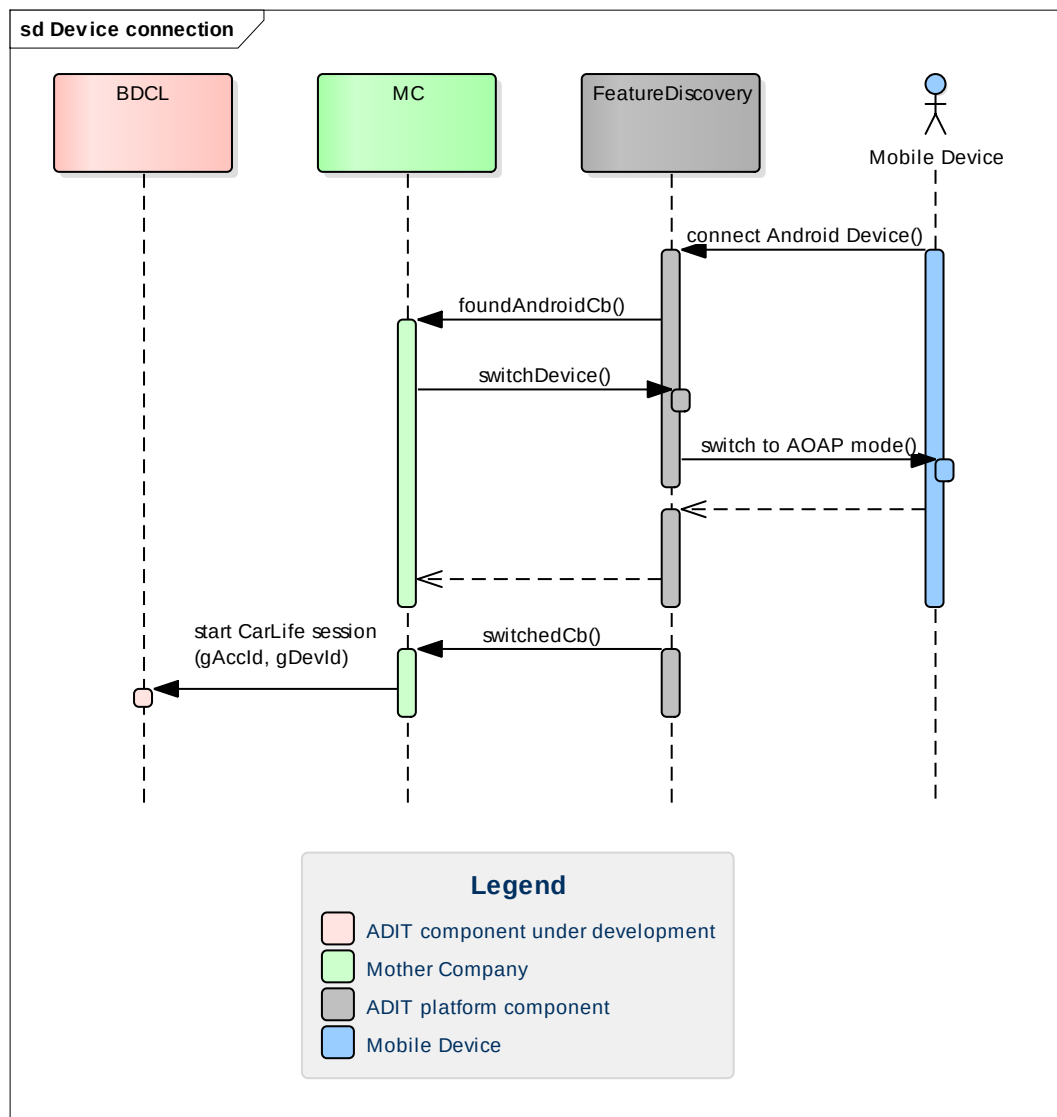


Figure 6 - Physical connection of Android device

4.4.2 Physical Connection of iOS Device

Figure 7 how iOS device is detected and switched to host mode using the FeatureDiscovery and iAP2 component. To launch BDCL in iOS device HU must be in gadget mode and Apple device should be in host mode. MC application has to start iAP2 session and configure to start EA Native Transport.

Requirement to start Baidu CarLife is that MC application has to provide WriteFD and ReadFD to aditCoreSurrogate once EA session started successfully.

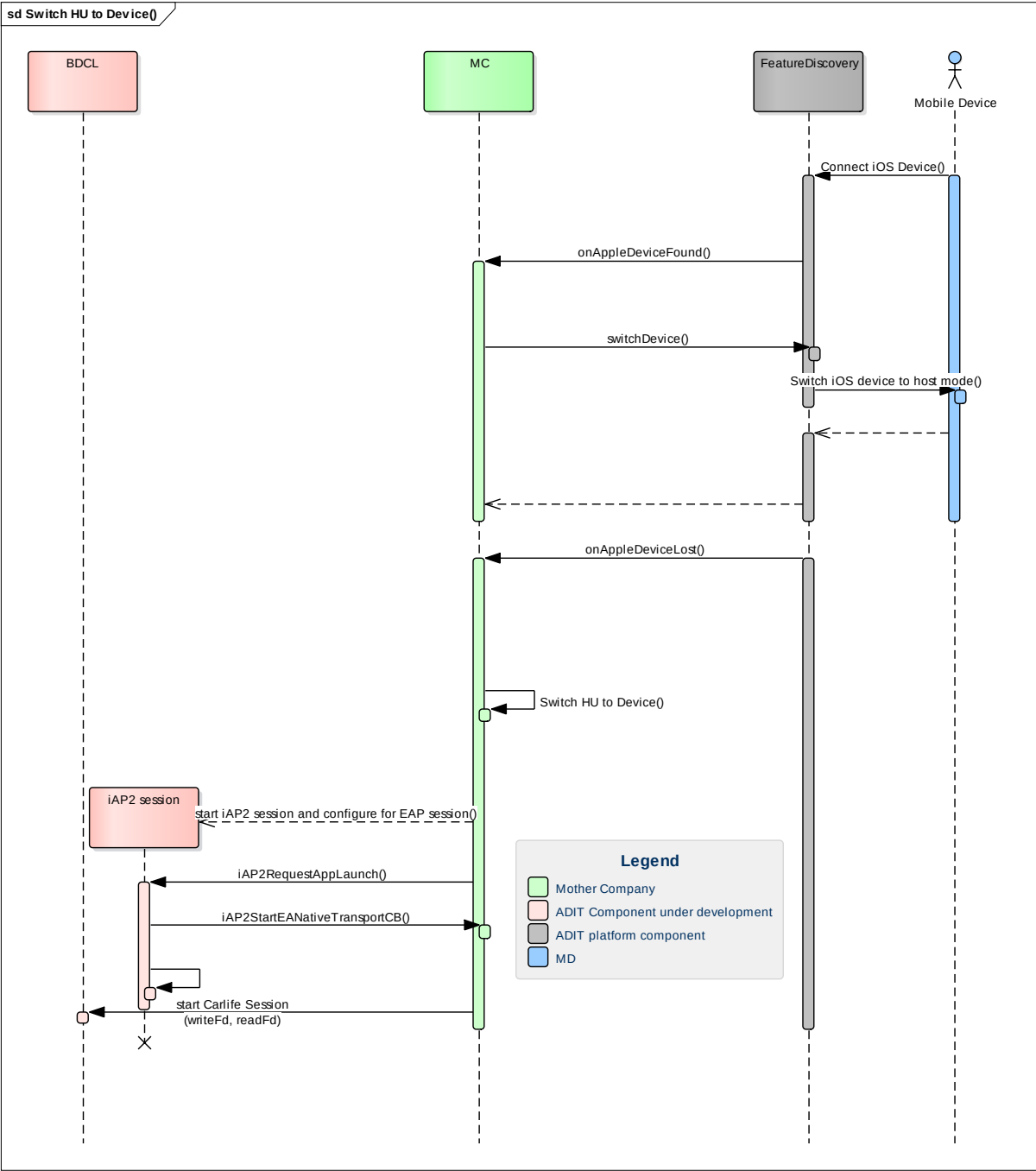
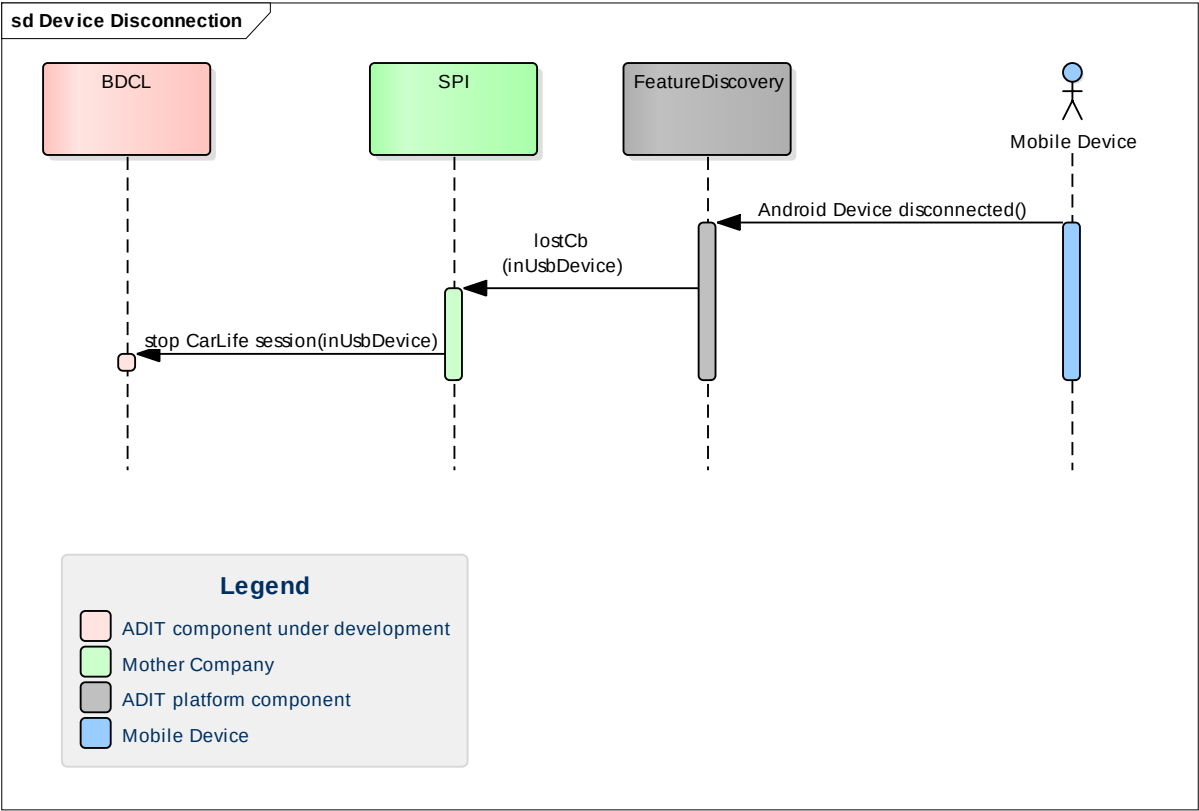


Figure 7 - Physical connection of iOS device

4.4.3 Physical Disconnection of Android Device

Figure 8 shows the physical disconnection of Android device and termination of BDCL session using FeatureDiscovery.

191



192

193

Figure 8 - Physical disconnection of Android device

194 **4.4.4 Physical Disconnection of iOS Device.**

195 Figure 9 shows the physical disconnection of iOS Device. Since HU will be in gadget mode
196 FeatureDiscovery will not receive any notification about the iOS device disconnection. MC application
197 will get the iOS device disconnection notification through iAP2 session.

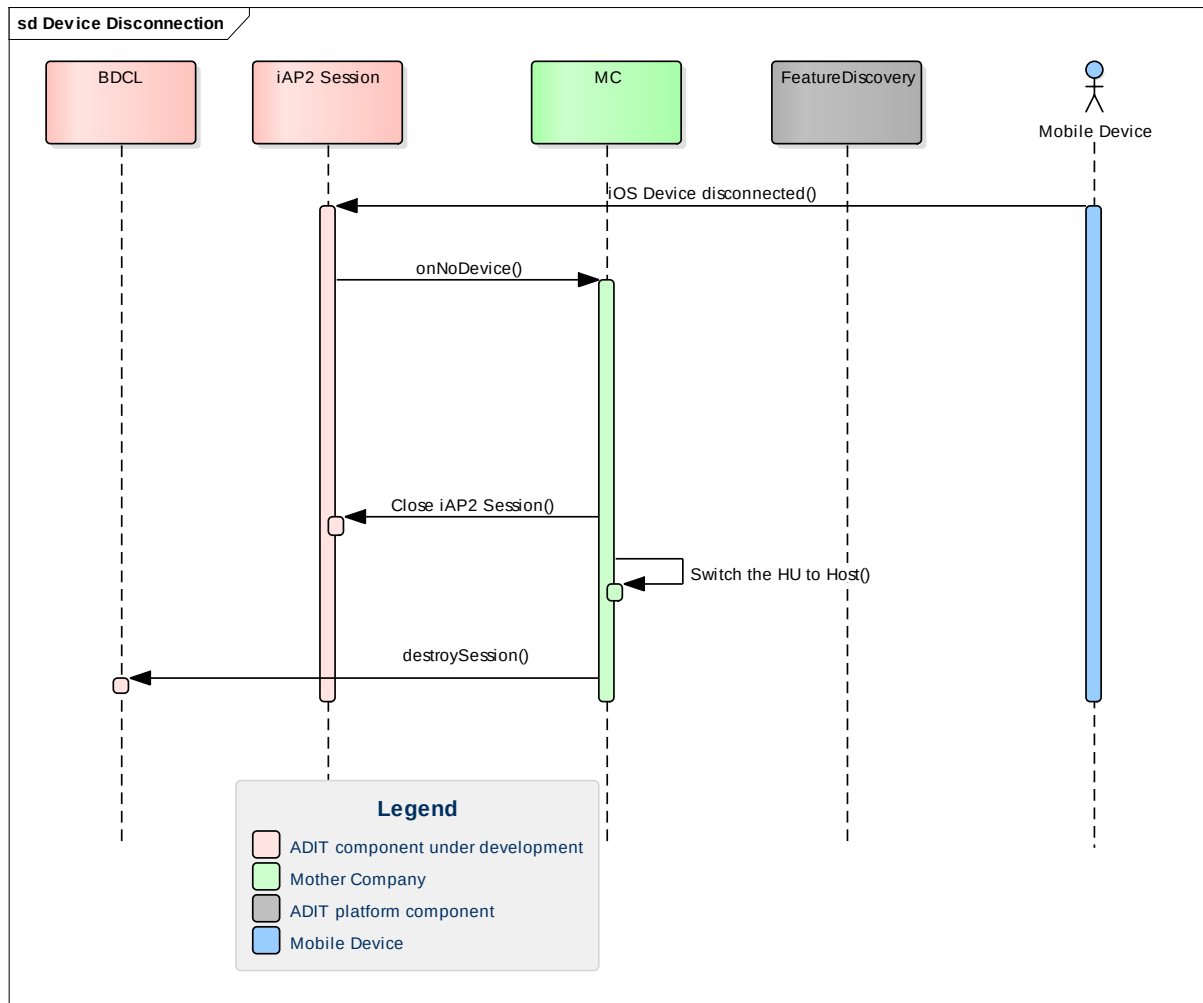


Figure 9 - Physically disconnection of iOS device

5 Design

5.1 Transport Handling

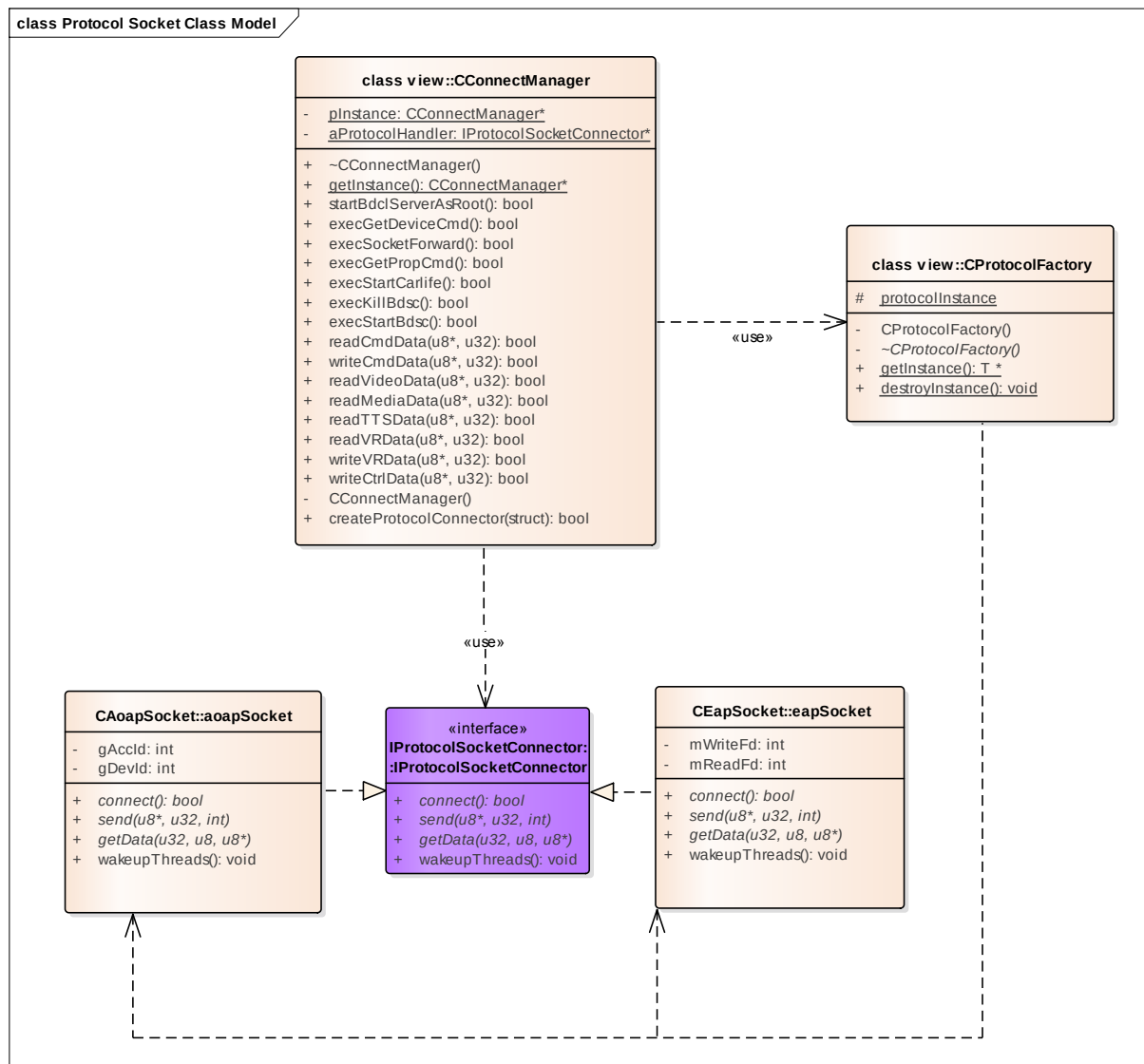


Figure 10 – Transport Handling Class Model Overview

Figure 10 gives the insight of modification done on Baidu Core. The Code originally given by the Baidu was creating 6 TCP/IP sockets which in turn was communicating with either AOAP/EA Native Transport. But that has been modified as shown in the above figure by bypassing the TCP/IP sockets. Now the instance of either AOAP or EA Native Transport is created based on the MD connected to communicate directly with underlying transport protocol (AOAP or EA Native Transport). createProtocolConnector creates the instance the either aoapSocket or eapSocket class depends on the attributes passed to this function, with the help of CProtocolFactory class.

Both aoapSocket and eapSocket class inherits common interface class IProtocolSocketConnector. CConnectManager will create an instance of IProtocolSocketConnector to send and receive the packets.

6 Interfaces for BDCL Platform

6.1 Adapter Interfaces Overview

This chapter explains the application interfaces with platform components and also with Baidu core as shown in Figure 11.

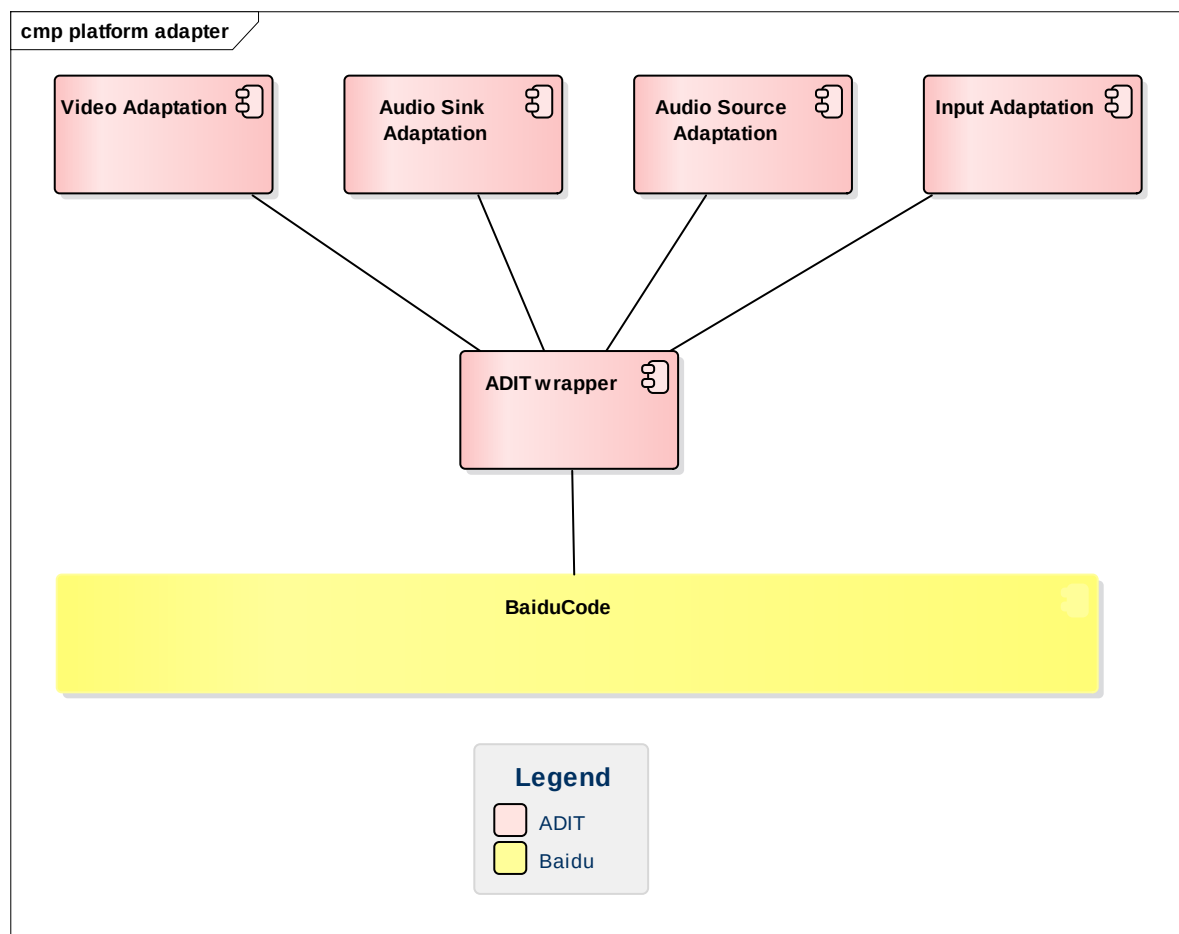


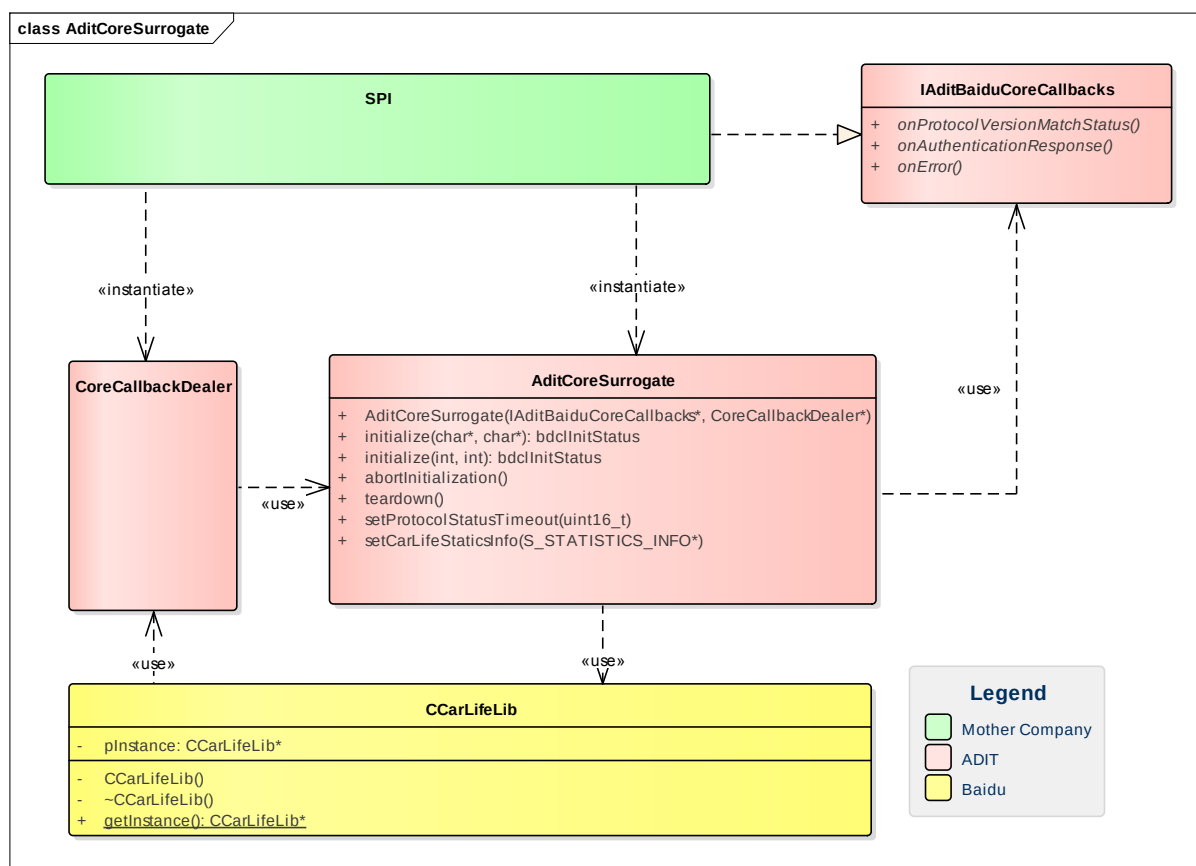
Figure 11 – Baidu Platform Adaptation

6.2 Adit Wrapper Adaptation

Figure 12 shows the relationship between MC application and ADIT wrapper.

ADIT wrapper component is a layer between the Baidu core and the MC application which initializes the CarLife library, which is responsible for creating the transport component depends on type of phone connected. Also ADIT wrapper component communicates the first handshaking message to MD (protoMatchStatus) and provides the authentication information of the HU through CarlifeStatisticInfo message and also provides a proxy mechanism to connect the callbacks from the Baidu core to the platform adaptations. This way the platform adaptations are decoupled from the Baidu stack and can be easily exchanged.

230



231

232

Figure 12 - ADIT Wrapper Overview

233 Application has to create the instance of CoreCallbackDealer and AduitCoreSurrogate.
 234 CoreCallbackDealer class handles all the static callback from the Baidu core whereas
 235 AduitCoreSurrogate takes care of initializing the Baidu library.

236 AduitCoreSurrogate::Initialize is overloaded by function parameter. Based on the phone type, MC
 237 application has to call the respective AduitCoreSurrogate::Initialize function. In case of Android phone,
 238 AduitCoreSurrogate::Initialize (int aoapAccessoryId, int aoapDeviceId) must be called. When as iOS
 239 device connected, AduitCoreSurrogate::initialize(const char* wFile, const char* rFile) should be called.

240 Calling AduitCoreSurrogate::initialize blocks until initialization finished or MC application aborts the
 241 initialization using AduitCoreSurrogate::abortInitialization().

242 IAditBaiduCoreCallbacks is an interface provided by ADIT. The methods are used to inform MC
 243 application on protocol version match, authentication response, error so that MC application can react
 244 accordingly. onError is called when an error occurs in an underlying layer.

245

246 6.2.1 Baidu CarLife Initialization.

247 This chapter explains about Initialization and termination of Baidu CarLife session for Android and iOS
 248 device.

249 Table 1 shows the messages and callbacks from Baidu core that are registered by the ADIT. MC has to
 250 register for remaining callbacks from Baidu core other than mentioned in the above table. Please refer
 251 Baidu CarLife Integration Specification for remaining callbacks.

Message Type	Callback from Stack
cmdRegisterProtocolVersionMatchStatus	MSG_CMD_PROTOCOL_VERSION_MATCH_STATUS
cmdRegisterMdAuthenResponse	MSG_CMD_MD_AUTHEN_RESPONSE
cmdRegisterVideoEncoderInitDone	MSG_CMD_VIDEO_ENCODER_INIT_DONE
cmdRegisterConnectException	MSG_CMD_CONNECT_EXCEPTION
videoRegisterDataReceive	MSG_VIDEO_DATA
mediaRegisterInit	MSG_MEDIA_INIT
mediaRegisterNormalData	MSG_MEDIA_DATA
mediaRegisterStop	MSG_MEDIA_STOP
mediaRegisterPause	MSG_MEDIA_PAUSE
mediaRegisterResume	MSG_MEDIA_RESUME_PLAY
ttsRegisterInit	MSG_NAVI_TTS_INIT
ttsRegisterNormalData	MSG_NAVI_TTS_DATA
vrRegisterNormalData	MSG_VR_AUDIO_DATA
vrRegisterStop	MSG_VR_AUDIO_STOP
vrRegisterInit	MSG_VR_AUDIO_INIT
vrRegisterInterrupt	MSG_VR_INTERRUPT

cmdRegisterMicRecordWakeupStart	MSG_CMD_MIC_RECORD_WAKEUP_START
cmdRegisterMicRecordEnd	MSG_CMD_MIC_RECORD_END
cmdRegisterMicRecordRecogStart	MSG_CMD_MIC_RECORD_RECOG_START

Table 1 - Messages consumed by ADIT

6.2.1.1 Baidu CarLife Session Initialization for Android Device.

Figure 13 shows the sequence diagram for initializing the CarLife session for Android device. HU uses AOAP communication to send and receive the messages to MD. AOAP receiver thread reads the data from MD and writes to respective ring buffer based on the channel ID.

Baidu core has 5 ring buffer to buffer the data for each channel (cmd, video, media, VR, Navi TTS) and also contains 5 receiver threads for each channel which reads the data from each ring buffers and provides it to protobuf.

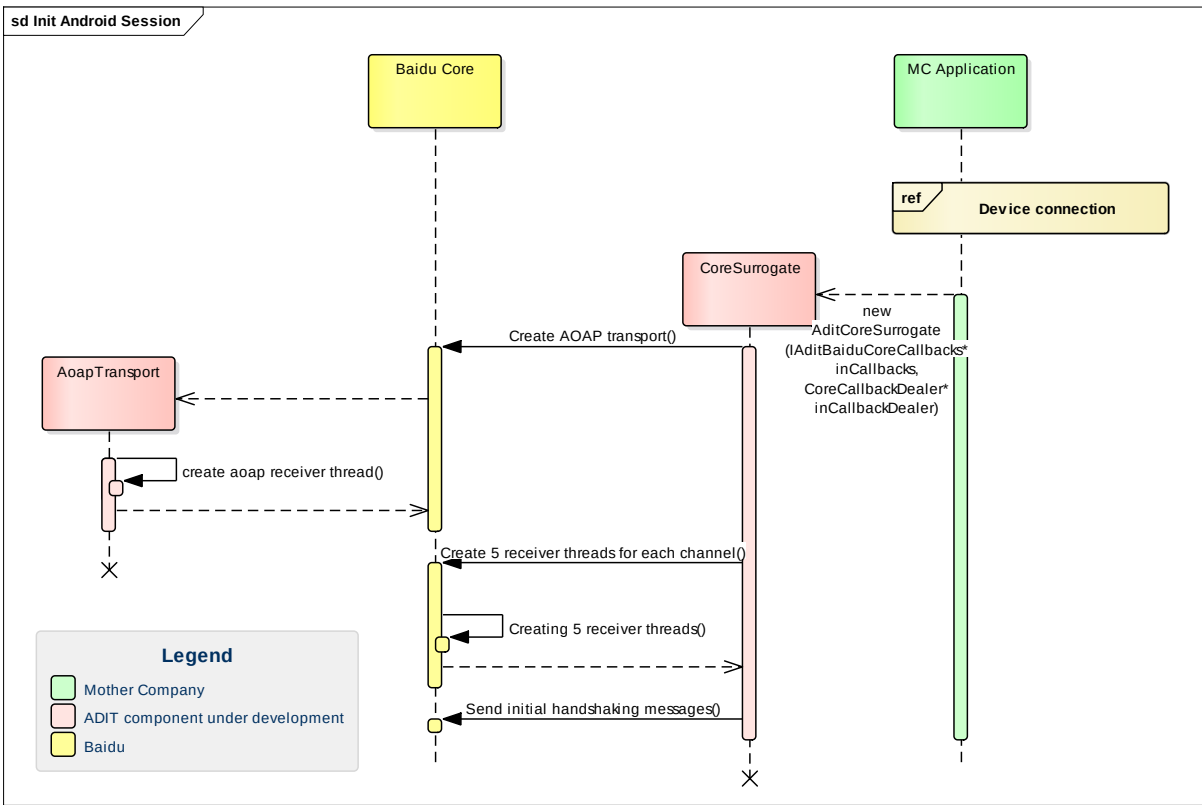


Figure 13 - Baidu CarLife session initialization for Android Device

6.2.1.2 Baidu CarLife session Initialization for iOS Device.

Figure 14 shows the sequence diagram for initializing the CarLife session for iOS device. HU uses EA Native Transport to send and receive the messages to MD. EA Native Transport receiver thread reads the data from MD and writes to respective ring buffer based on the channel ID.

Baidu core has 5 ring buffer to buffer the data for each channel (cmd, video, media, VR, Navi TTS) and also contains 5 receiver threads for each channel which reads the data from each ring buffers and provides it to protobuf.

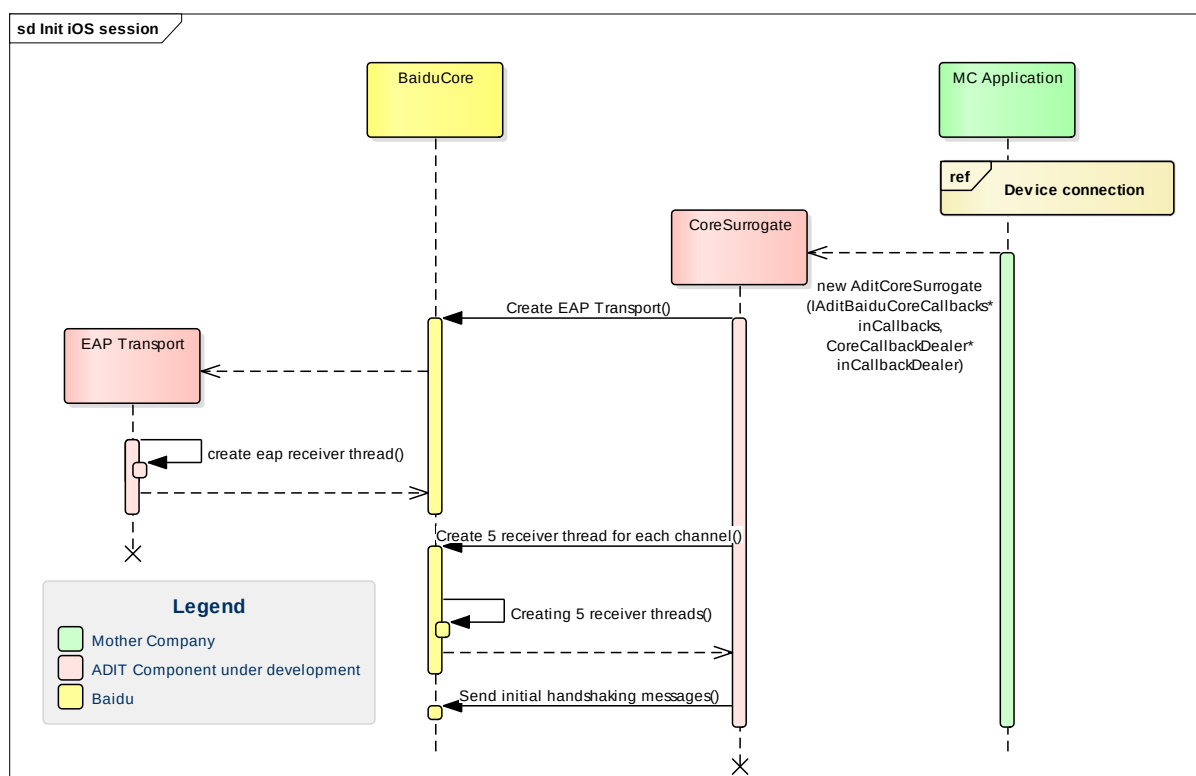


Figure 14 - Baidu CarLife session initialization for iOS device

6.2.1.3 Baidu CarLife session termination for Android Device.

Figure 15 shows the termination of Baidu CarLife session for Android device. On device disconnection application has to call teardown() which gracefully closes all the 5 receiver thread and closes the AOAP transport session. And de initializes the Carlifelib.

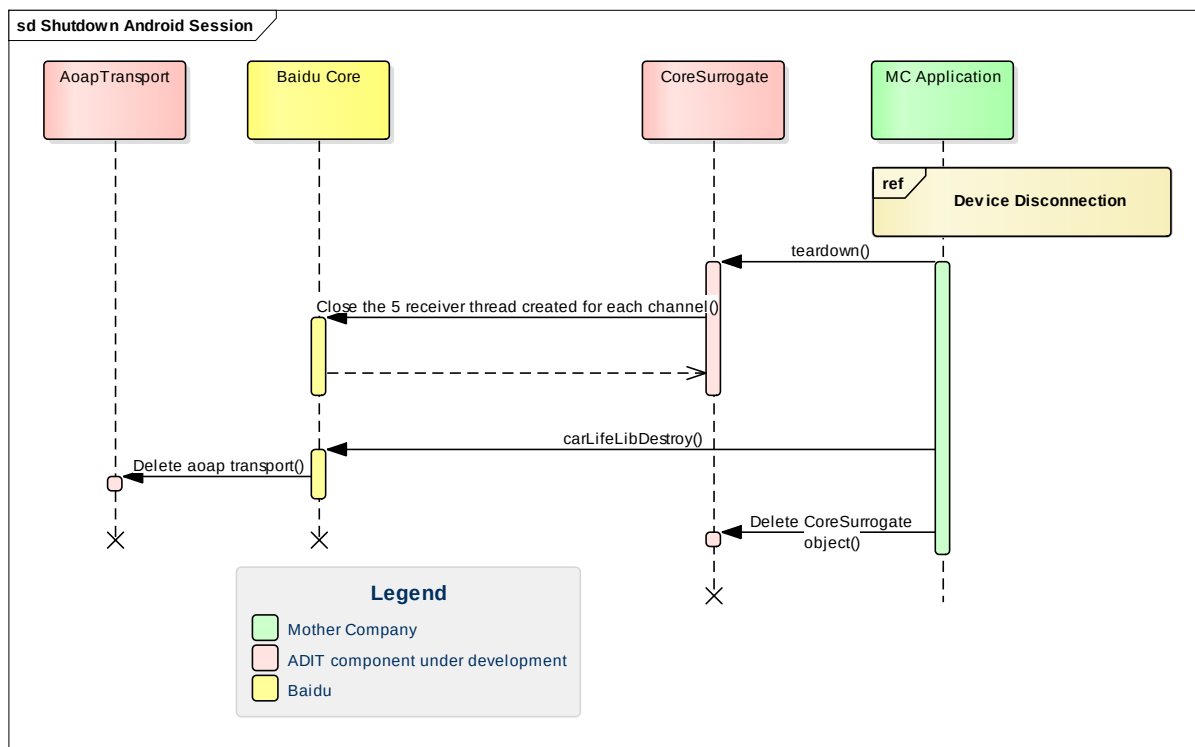


Figure 15 - Baidu CarLife session termination for Android Device

6.2.1.4 Baidu CarLife session termination for iOS Device.

Figure 16 shows the termination of Baidu CarLife session for iOS device. On device disconnection MC application has to call `teardown()` which gracefully closes all the 5 receiver thread and closes the EA Native Transport session and de initializes the Carlifelib.

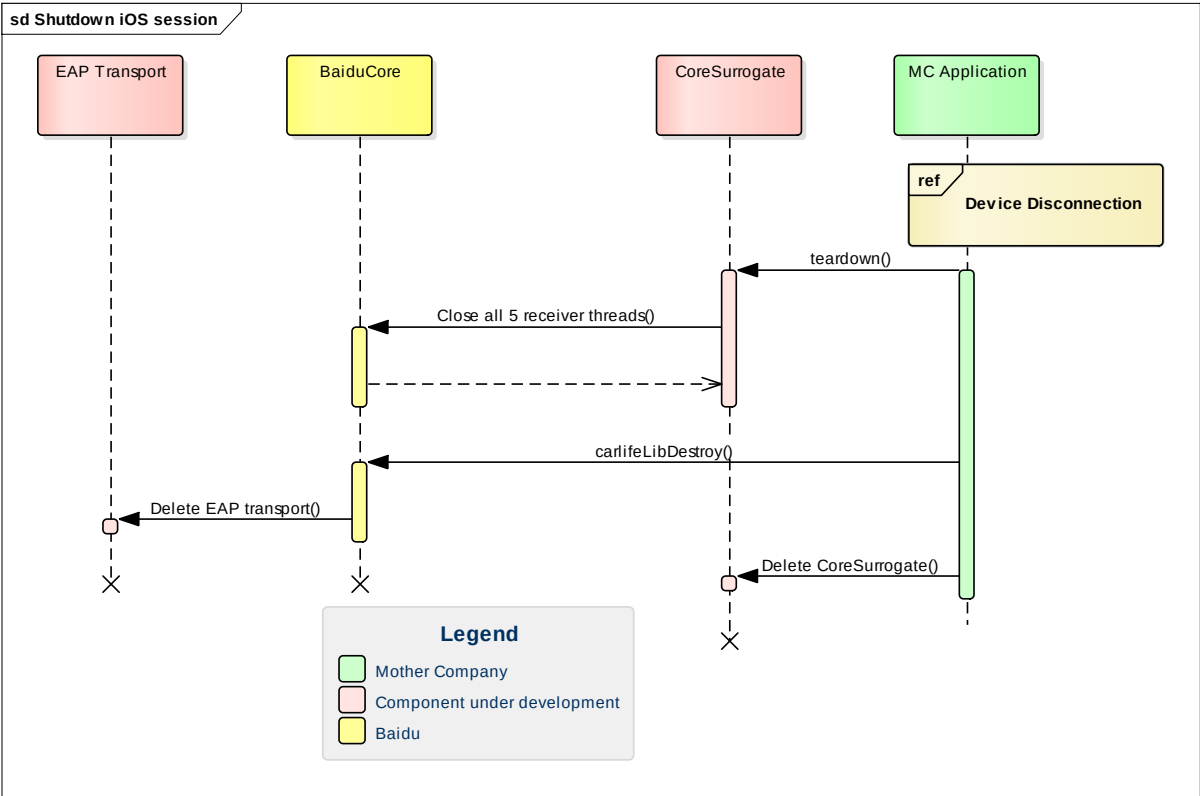


Figure 16 - Baidu CarLife session termination for iOS device

6.3 Video Sink Adaptation

The Video Adaptation handles the video frames received from the mobile device to project them in the display and provides two types of interface: AditVideoSink and IAditVideoSinkCallbacks. Figure 17 shows the relationship between MC application and Video Sink endpoint.

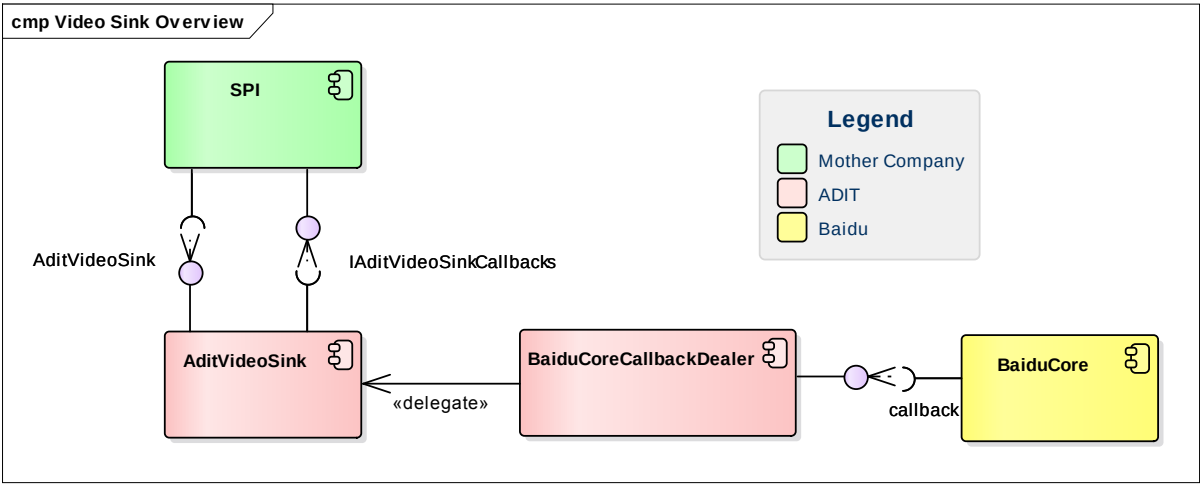


Figure 17 - Video Sink Overview

AditVideoSink class is provided by ADIT and defines the interface between MC application and Video Sink Endpoint. The purpose of AditVideoSink is to avoid interface changes in future, in case Video Sink will be implemented using another technology than GStreamer. BaiduCoreCallbackDealer acts as delegator, which passess received video packets to AditVideoSink.

Figure 18 shows the structure of Video Sink endpoint.

GstreamerVideoSink inherits AditVideoSink, GstreamerVideoSink handles video data internally.

IAditVideoSinkCallbacks is an interface provided by ADIT. Its methods are used to inform MC application on first frame rendered, connection exception, error or video encoder init done, so that MC application can react accordingly. onError is called when an error occurs in an underlying layer.

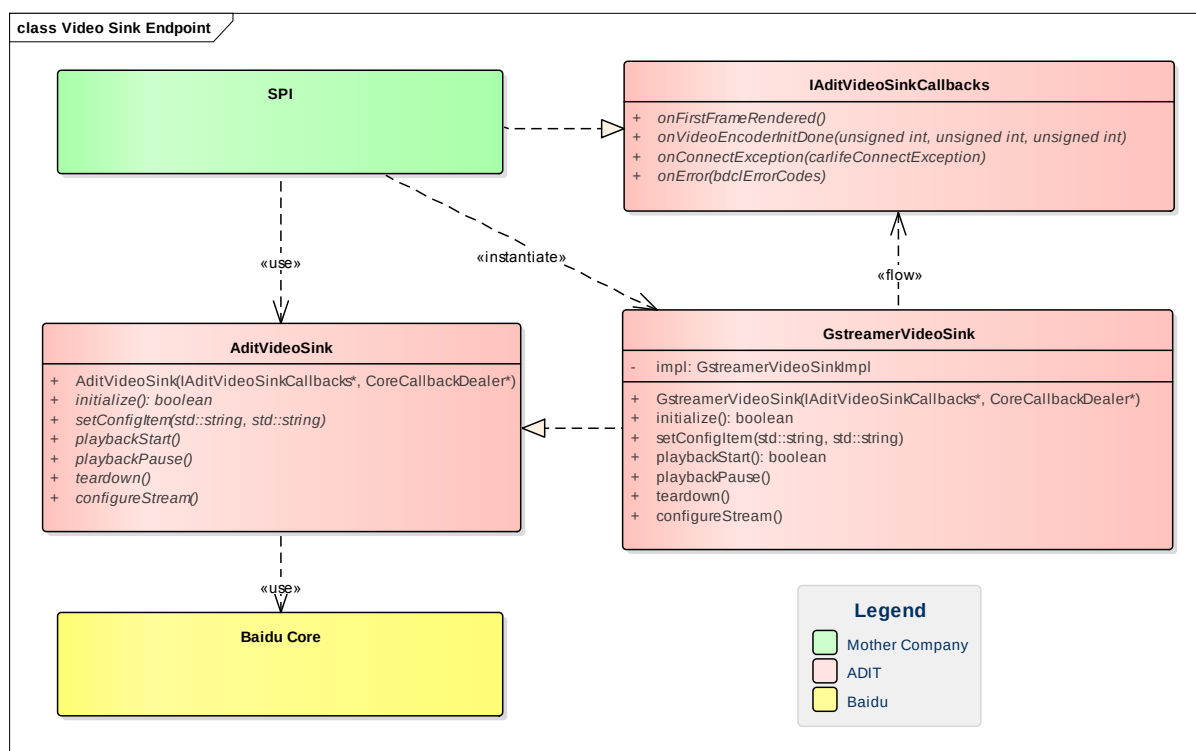


Figure 18 - Adit Video Adaptation

6.3.1 Video Sink Configuration

Table 2 shows configuration parameters for Video Sink endpoint. All parameters have to be configured using setConfigItem method of AditVideoSink class.

Key name	Description
gstreamer-video-pipeline	[mandatory] GStreamer video pipeline to be created. The appsrc element will be attached to the front of the pipeline.

	(e.g." vpudec low-latency=true frame-plus=2 framedrop=false framerate-nu=60 dis-reorder=true ! gst_apx_sink display-width=960 display-height=540 layer-id=3000 surface-id=40 sync=false qos=false max-lateness=3000000000 ")
video-width	[optional, default=800] Number of pixels in the x axis for a logical UI resolution
video-height	[optional, default=480] Number of pixels in the x axis for a logical UI resolution
video-fps	[optional, default=30] Frame rate of video
gststreamer-videosink-record	This feature still not available.

Table 2 - Configuration for Video Sink Endpoint

6.3.2 Video Adaptation Initialization

Figure 19 shows the initialization of Video Sink Endpoint.

AditVideoSink class instance is created by MC application and sets configuration data for all necessary configuration items (please refer to Table 2 - Configuration for Video Sink Endpoint). Before creating the instance of AditVideoSink, MC has to make sure to create an instance of CoreCallbackDealer class.

On calling configureStream() MD initializes its video encoder and reply back with negotiated display resolution and framerate. MC has to set received configuration values again. Once MC application owns the screen resource then MC can call playbackStart(). By this time MC application needs to be fully prepared for video playback, because MD will start streaming the video packets immediately on receiving cmdVideoEncoderStart(). onFirstFrameRenderedCallback is called when the first frame is rendered to the screen.

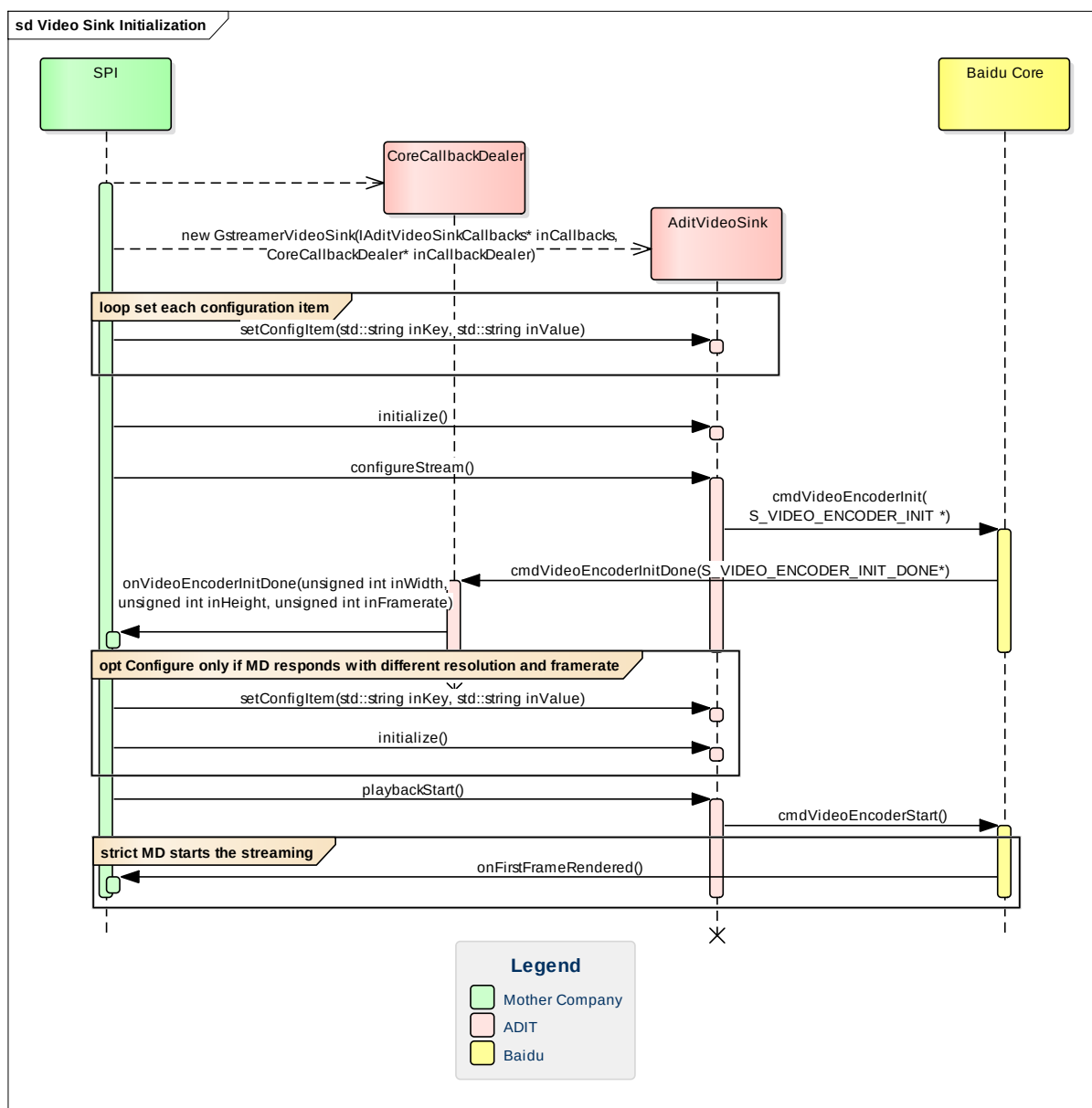


Figure 19 - Video Sink Initialization

For Gen3 platform two types of video Sink implementations available.

1. GstreamerVideoSink
2. UspiVideoSink

Whereas for Gen4 platform only UspiVideoSink can be used. Interface remains the same across two implementations.

UspiVideoSink uses USPI::Video backend, which in turn takes care of Gstreamer versions.

6.4 Audio Sink Adaptation

There are three audio stream type (ex. Media, VR TTS , Navi TTS) and each stream type is represented by an Audio Sink endpoint.

The Audio Sink Adaptation handles the audio packets received from the mobile device to stream them to the underlying audio hardware. It also buffering the audio data to compensate for network jitter and allowing the audio streaming to be delayed while the phone has already started to stream the audio data.

Figure 20 shows the relationship between MC application and Audio Sink Adaptation.

AilAudioSink class is provided by ADIT and defines the interface between MC application and Audio Sink Endpoint.

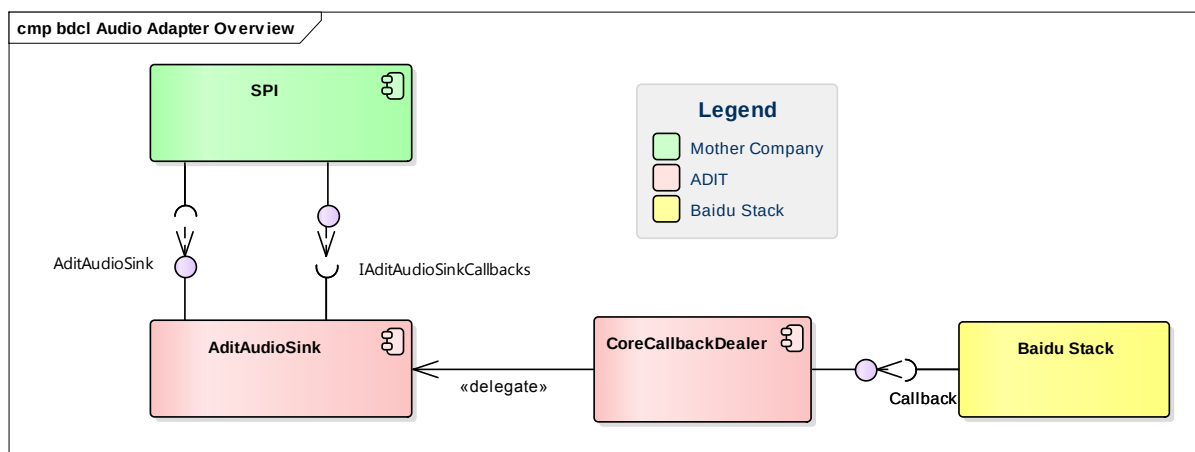


Figure 20 - Audio Sink Overview

AditAudioSink class is provided by ADIT. This class defines the interface between MC application and Audio Sink endpoint. The purpose of AditAudioSink is to avoid interface changes in future, in case Audio Sink will be implemented using another technology than ALSA.

Figure 21 shows the structure of Audio Sink endpoint.

AilAudioSink inherits AditAudioSink, AilAudioSink handles the audio data internally.

IAditAudioSinkCallbacks is an interface provided by ADIT. Its methods are used to inform MC application on playback init, playback stop, playback resume, playback pause, playback interrupt or error, so that MC application can react accordingly. BaiduCoreCallbackDealer acts as delegator, which passes received video messages to AditVideoSink.

onError is called when an error occurs in an underlying layer.

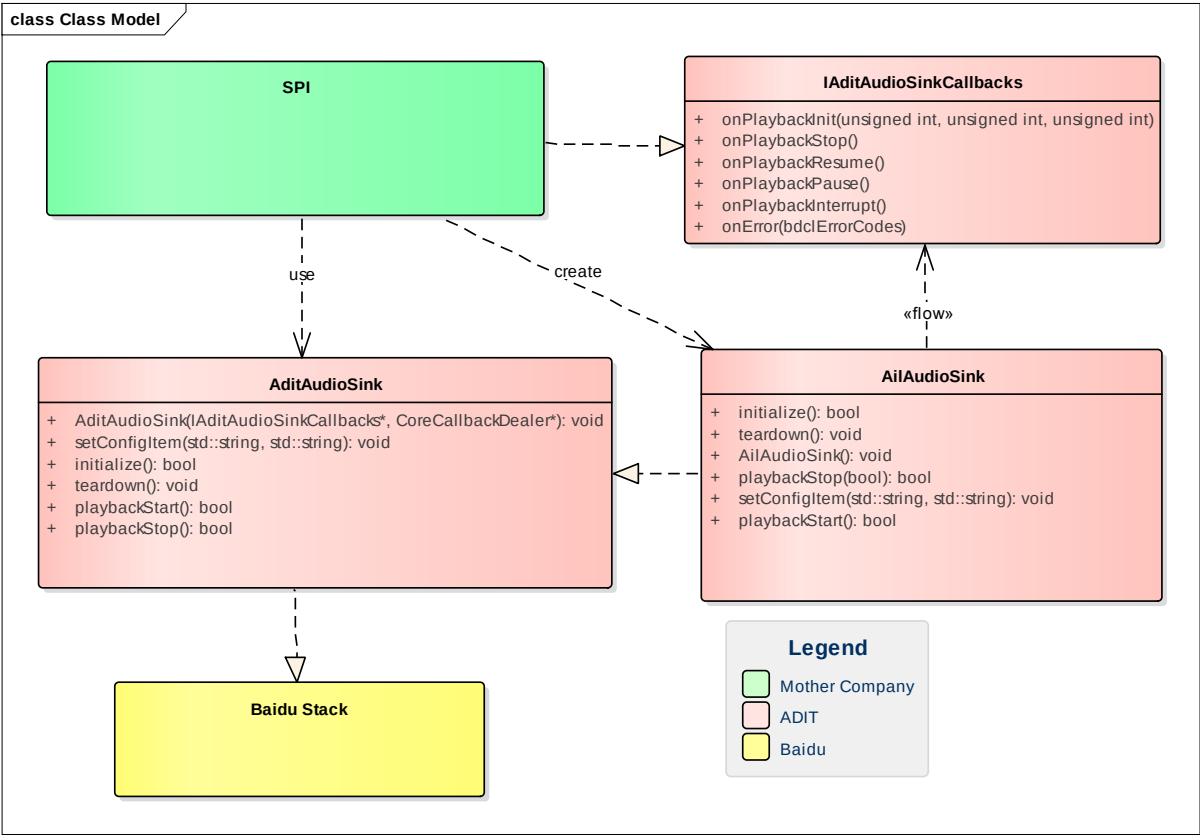


Figure 21- bdcl Audio Sink Adaptation

6.4.1 Audio Sink Configuration

Table 3 shows configuration parameters for Audio Sink endpoint (Media/Navi/VR). All parameters have to be configured using setConfigItem method.

Key name	Description
audio-playback-device	Multikey configuring: - Device: Name of Alsa device [mandatory] - periodMs: Duration of one period [optional, default=8] - inittoutms: Initial time out for AIL to start processing callback [optional, default=500ms] ex: device=entertainment_main periodMs=8 inittoutms=2000
audio-sink-stream-type	[mandatory] Stream type: AUDIO_STREAM_MEDIA AUDIO_STREAM_NAVI_TTS

	AUDIO_STREAM_VR_TTS
audio-sink-sampling-rate	[optional, default=16000] Sampling rate: 16000 44100 According to Baidu Spec: NAVI TTS, VR TTS, VR MIC have 16kHz; MEDIA has 44.1kHz per default
audio-sink-bits-per-sample	[optional, default=16] Number of bits per sample. According to Baidu Spec: NAVI TTS, VR TTS, VR MIC, MEDIA have 16 bits per sample.
audio-sink-channels	[optional, default=1] Number of channels: 1 2 According to Baidu Spec: NAVI TTS, VR TTS, VR MIC have mono channel; MEDIA has stereo channel per default
disable-real-time-priority-audio	[optional, default=0] Flag to manually disable real-time priority in audio threads. 0: enable 1: disable
audio-threads-real-time-priority	[optional, default=61] Real-time thread priority.
audio-sink-threshold-ms	[optional, default=100] Duration in milliseconds which shall be buffered additional to the calculated jitter (expected time interval between two audio data packages from the MD). If the buffer contains less data than configured, than audio will be started to play from the next available period which crossing the threshold.

audio-sink-buffer-max-ms	[optional, default=20000] Duration in milliseconds after which the half of the buffered data will be discarded. If MC application starts the audio stream late but if MD already started to send audio packets then half of the buffered data will be discarded once configured value is reached. Note: Default value is set based on Bosch MC's response. Please do set the value as mandatory based on your system performance. Just to make sure no packets will be discarded particularly in case of VR and Navi TTS.	
audio-sink-record		Flag to record Audio Sink audio data. Create/open file indicated by <i>filename</i> at playbackStartCallback. Write audio data in dataAvailableCallback. Create a WAV header, write the header to the recorded file and close the file at playbackStopCallback.
	enable	[optional, default=0] Flag to record audio data.
	filename	Destination of recorded audio file. (e.g. "/tmp/MyMediaRecord") Each file gets a numbering (e.g. "/tmp/MyMediaRecord0.wav")

Table 3 - Configuration for Audio Sink Endpoint

6.4.2 Audio Sink Adaptation Initialization

Figure 22 shows the initialization of Audio Sink Endpoint.

AditAudioSink class instance is created by MC application and sets configuration data for all necessary configuration items (please refer to Table 3 - Configuration for Audio Sink Endpoint). Before creating the instance of AditAudioSink, MC has to make sure to create an instance of CoreCallbackDealer class.

On user presses, the play button MD sends the mediaInit() callback with MD compatible configuration values. MC has to set received configuration values again. Once MC application owns the audio resource then MC can call playbackStart(). Audio Endpoint will buffer the audio data, because MD will start streaming the audio packets immediately after sending the mediaInit() callback. MC can configure the maximum size of the buffer through audio-sink-buffer-max-ms config item.

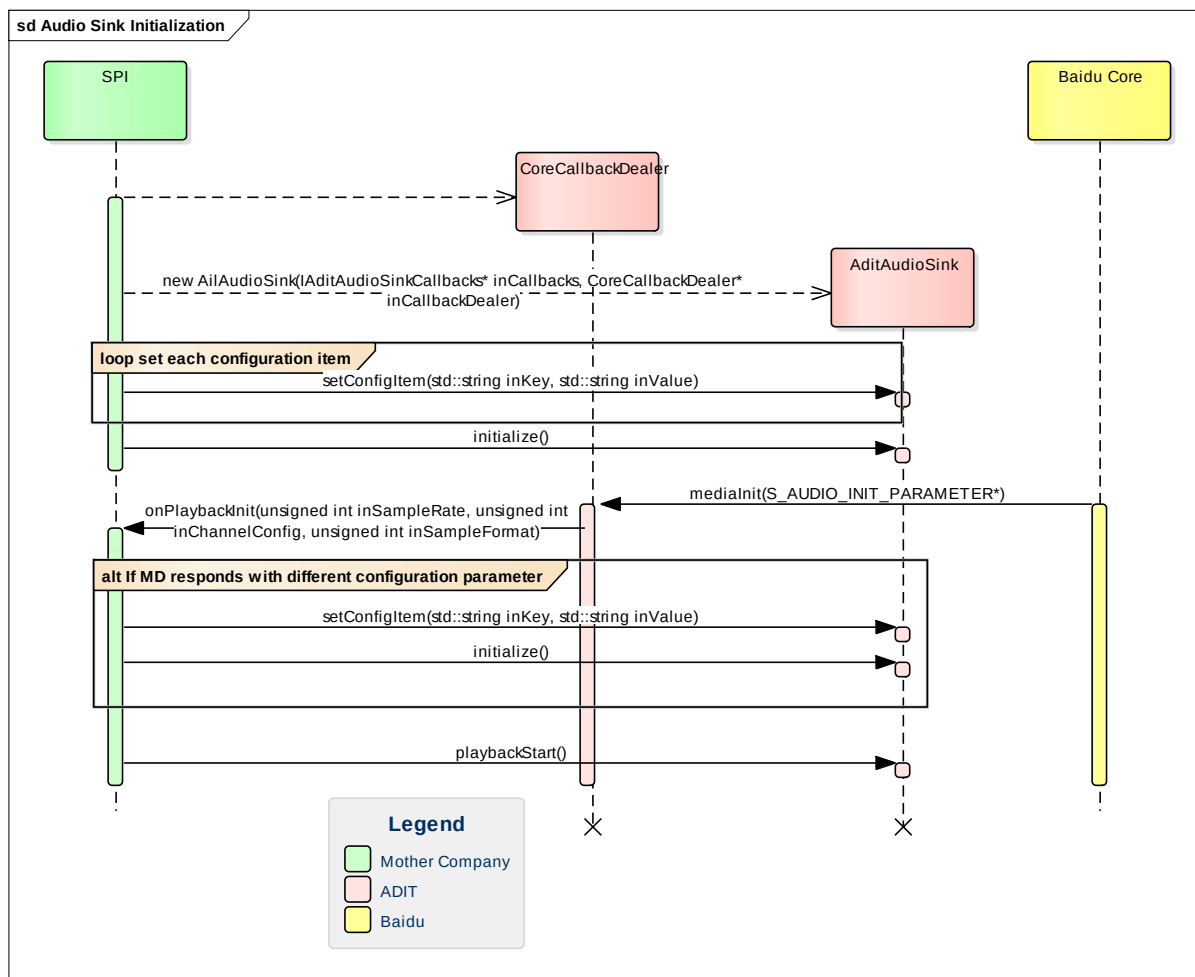


Figure 22 - Audio Adaptation Initialization

6.4.3 AIL Audio Sink Processing

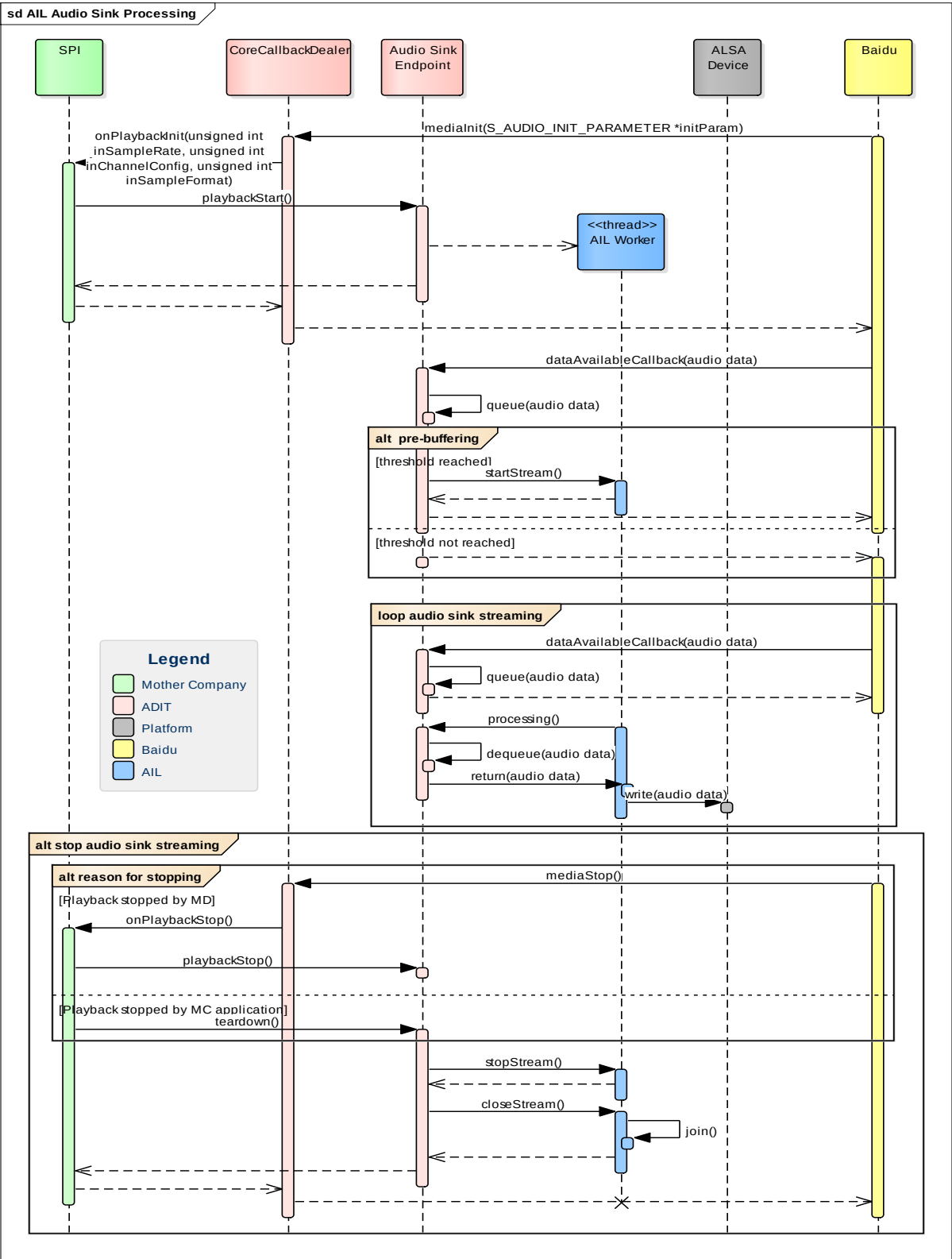
The Ail Audio Sink endpoint itself has no thread. AIL is responsible to process the incoming audio data.

Figure 23 shows creation, execution and destruction of the AIL Worker thread.

The AIL Worker thread is created when mediaInit is received from Baidu core and is only necessary while audio playback is running.

Baidu core informs Audio Sink endpoint by dataAvailableCallback about incoming audio data from MD. Audio Sink endpoint pushes the audio data to the internal queue to be processed. AIL Worker thread calls continuously the processing callback and available audio data are pulled from queue. AIL writes the provided audio data to ALSA.

387



388

389

Figure 23 - Ail Audio Sink processing

The AIL Worker thread is destroyed when the Baidu core sends a mediaStop/mediaPause.

MC application can configure whether Audio Sink endpoint should drain or drop the queued audio data when Baidu core sends a mediaStop/mediaPause (see Table 3 - Configuration for Audio Sink Endpoint).

The remaining audio is dropped regardless of the audio type in case the destruction occurs due to an ending session.

6.5 Audio Source Adaptation

Figure 24 shows the relationship between MC application and Audio Source endpoint. Audio Source endpoint provides two types of interface: AditAudioSource and IAditAudioSourceCallbacks. Audio Source and Audio Sink have similar structures. Audio Source handles the recorded PCM data to send them to MD.

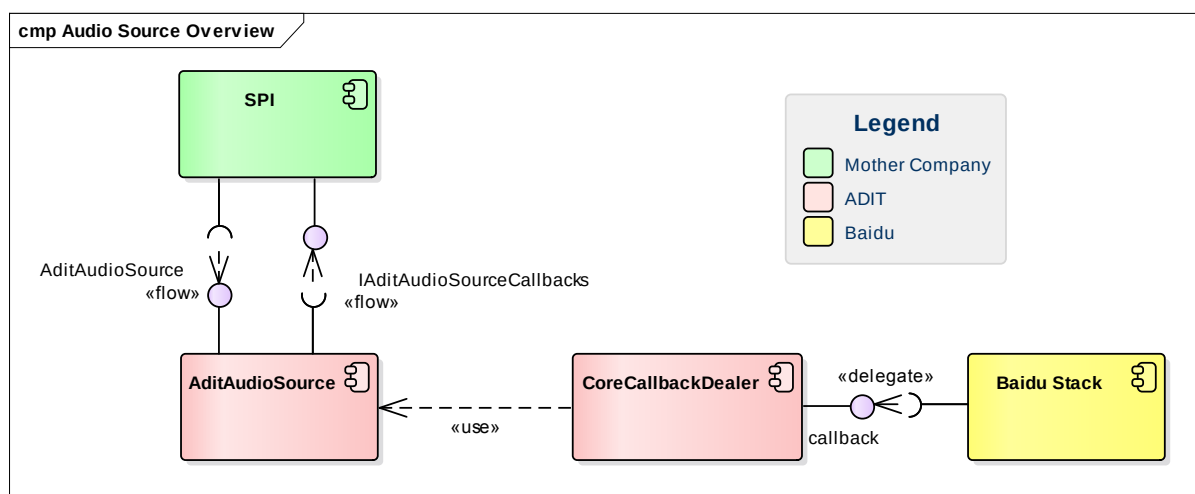


Figure 24 - Audio Source Overview

AditAudioSource class is provided by ADIT. This class defines the interface between MC application and Audio Source endpoint. The purpose of AditAudioSource is to avoid interface changes in future, in case Audio Source will be implemented using another technology than ALSA.

Figure 25 shows the structure of Audio Source endpoint.

AilAudioSource inherits AditAudioSource, AilAudioSource handles the audio data internally.

IAditAudioSourceCallbacks is an interface provided by ADIT. Its methods are used to inform MC application on MicRecordWakeupStart, MicRecordEnd and MicRecordRecognitionStart or Error, so that MC application can react accordingly.

onError is called when an error occurs in an underlying layer.

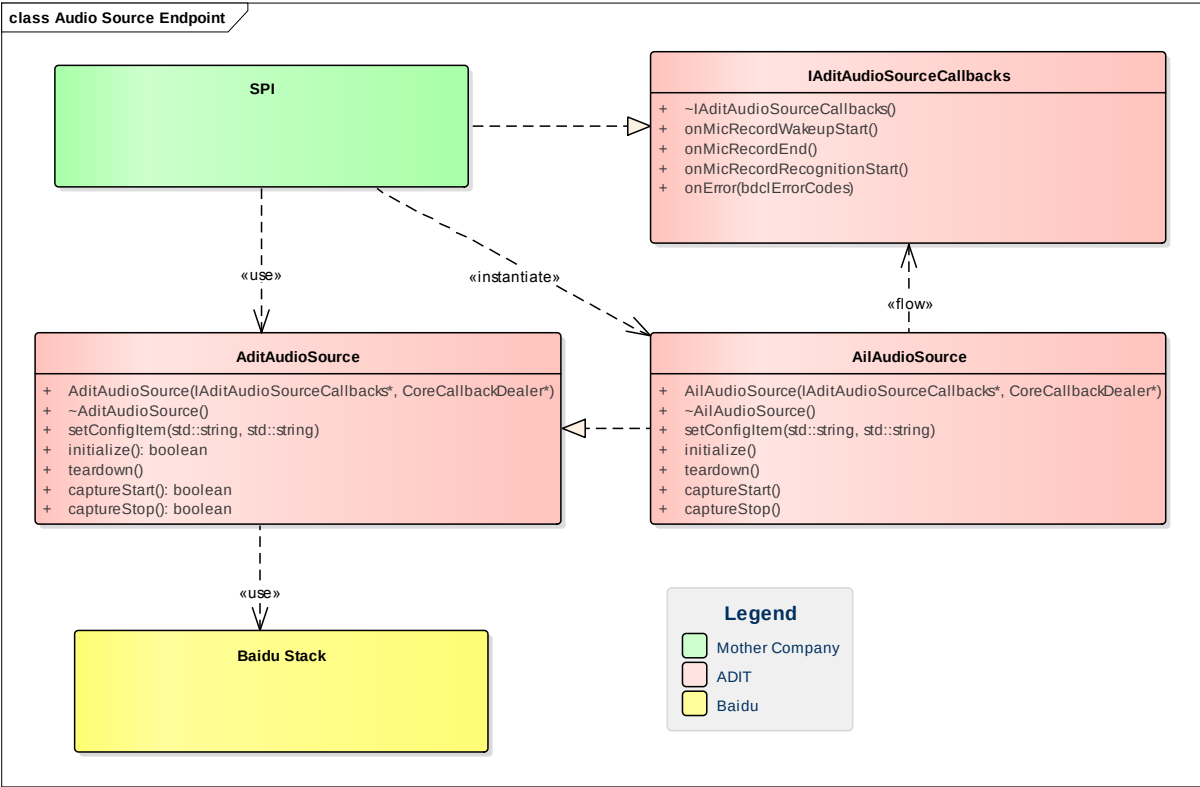


Figure 25 - bdcl Source Adaptation

6.5.1 Audio Source Configuration

Table 4 shows configuration parameters for Audio Source endpoint. All parameters have to be configured using setConfigItem method.

Key name	Description
audio-capture-device	Multikey configuring: 1. Device: Name of Alsa device [mandatory] 2. periodMs: Duration of one period [optional, default=8] 3. inittoutms: Initial time out for AIL to start processing callback [optional, default=500ms] ex: device=microphone periodMs=8 inittoutms=2000
audio-source-stream-type	[mandatory] Stream type: AUDIO_CAPTURE_VR_TTS
audio-source-sampling-rate	[optional, default=16000] Sampling rate:

	16000 According to Baidu Spec: VR CAPTURE has 16kHz.
audio-source-bits-per-sample	[optional, default=16] Number of bits per sample. According to Baidu Spec: VR CAPTURE has 16 bits per sample.
audio-source-channels	[optional, default=1] Number of channels: 1 According to Baidu Spec: VR CAPTURE has mono channel.
disable-real-time-priority-audio	[optional, default=0] Flag to manually disable real-time priority in audio threads. 0: enable 1: disable
audio-threads-real-time-priority	[optional, default=61] Real-time thread priority.

Table 4 - Audio Source Configuration

6.5.2 Audio Source Adaptation Initialization

Figure 26 shows the initialization of Audio Sink Endpoint.

AditAudioSource class instance is created by MC application and sets configuration data for all necessary configuration items (please refer to Table 4 - Audio Source Configuration). Before creating the instance of AditAudioSource, MC has to make sure to create an instance of CoreCallbackDealer class.

MD will notify in two ways to record the voice.

1. Voice Wakeup Interactive: In which user will provide the instruction through voice to start the recordings. This feature is not available in default; MC application has to enable this through MSG_CMD_MD_FEATURE_CONFIG_RESPONSE message.
2. Voice Recognition Interactive: In which recording will start once user presses the VR Icon on the screen.

Application has to call captureStart() to send the captured audio packets to Baidu core.

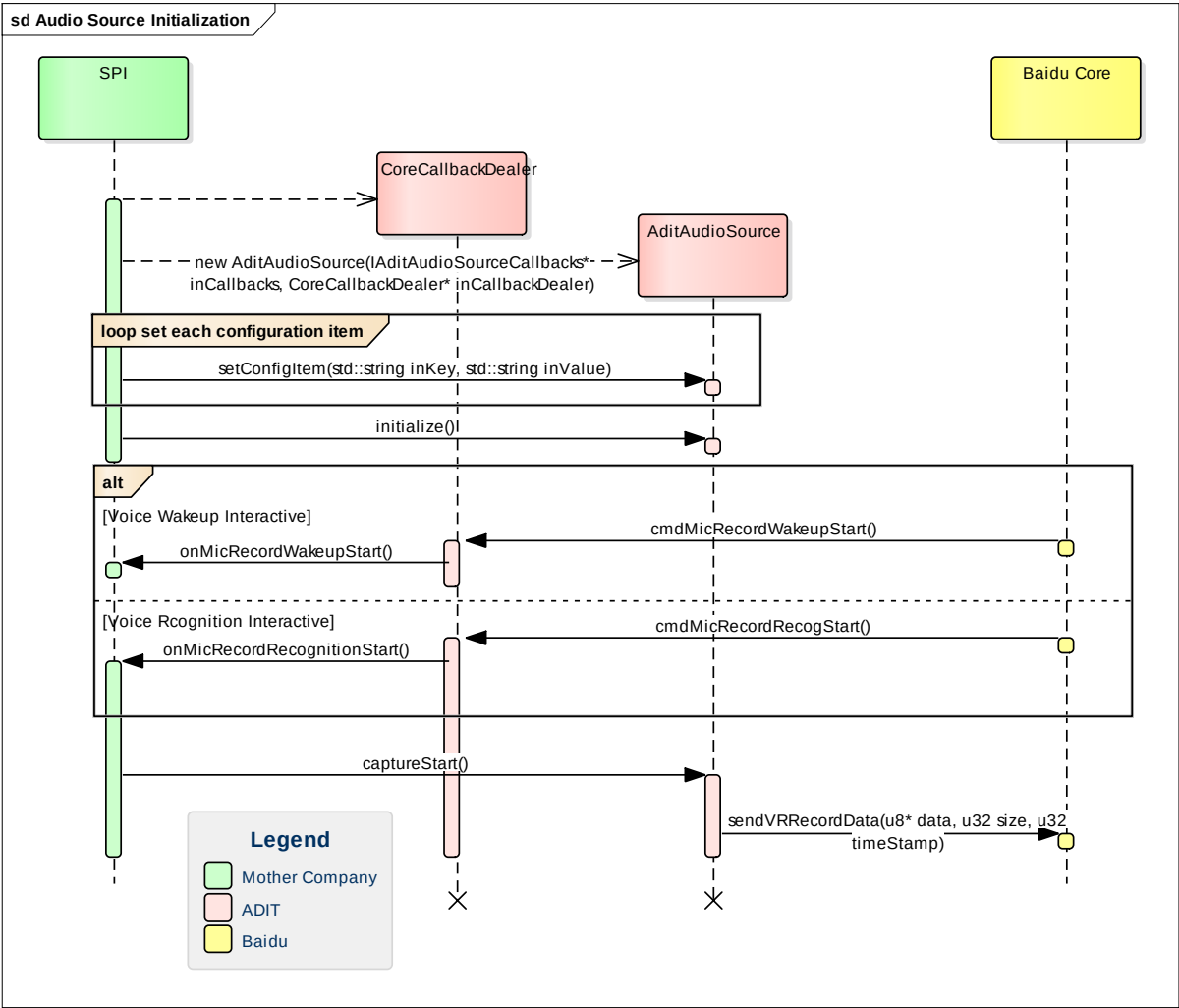


Figure 26 - Audio Source Adaptation Initialization

6.5.3 Audio Source Processing

The Ail Audio Source endpoint itself has no thread. AIL is responsible to process the audio data.

Figure 27 shows creation, execution and destruction of the AIL Worker thread.

The AIL Worker thread is created when a Mic Record request is received from Baidu core and is only necessary when audio capturing occurs.

When the AIL Worker thread is running, it captures the audio data from the audio library and exchange the audio data using processing callback functionality to AIL Audio Source endpoint continuously. AIL Audio Source endpoint copy the received audio data and forwards them to Baidu core.

The AIL Worker thread is destroyed when capturing is done and a Mic End request occurs or when the session is ending.

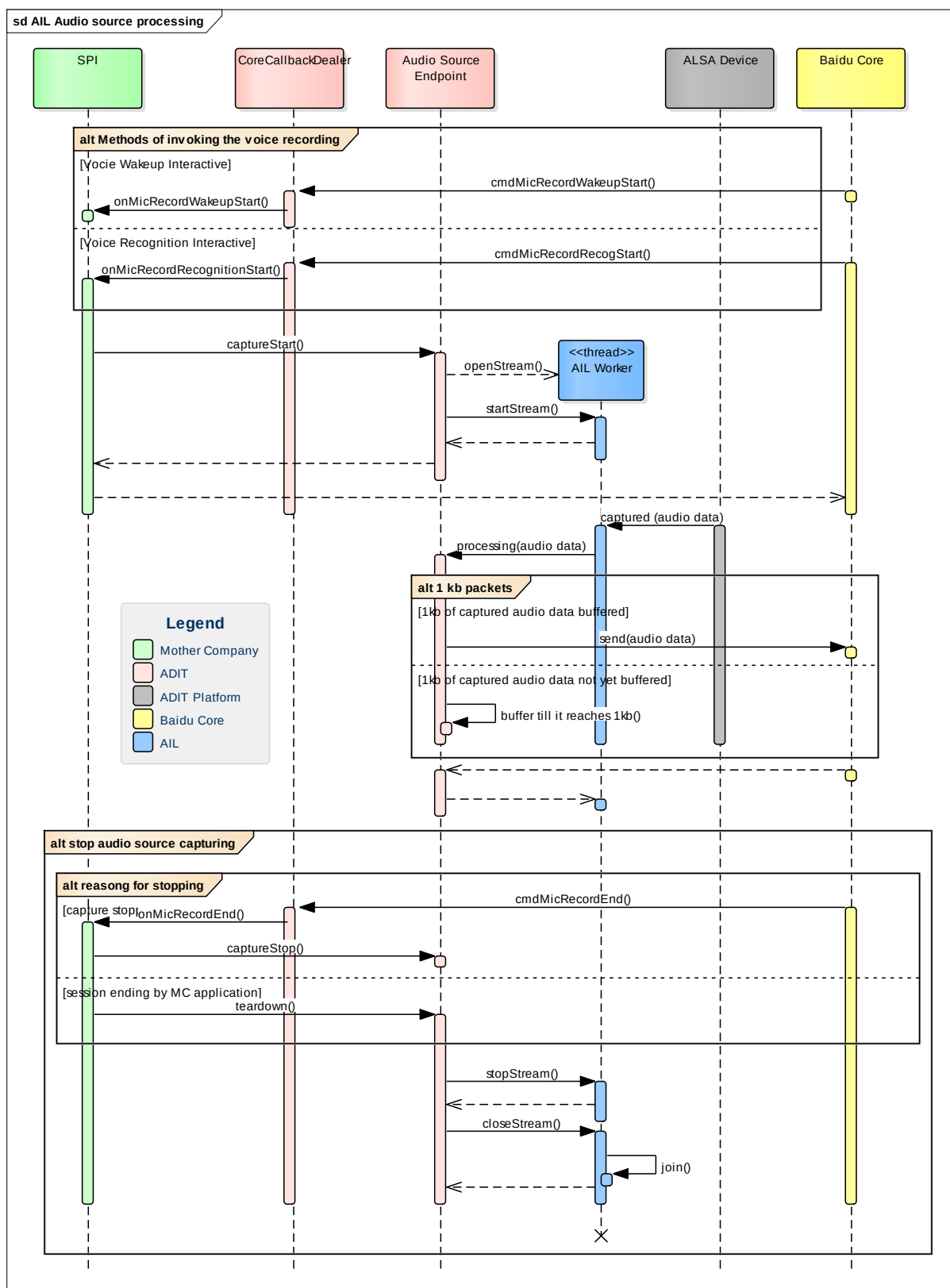


Figure 27 - Audio Source Processing

6.6 Input Source Adaptation

Figure 28 shows the relationship between MC application and Input Source endpoint. Input Source endpoint provides two interfaces: AditInputSource and IAditInputSourceCallbacks.

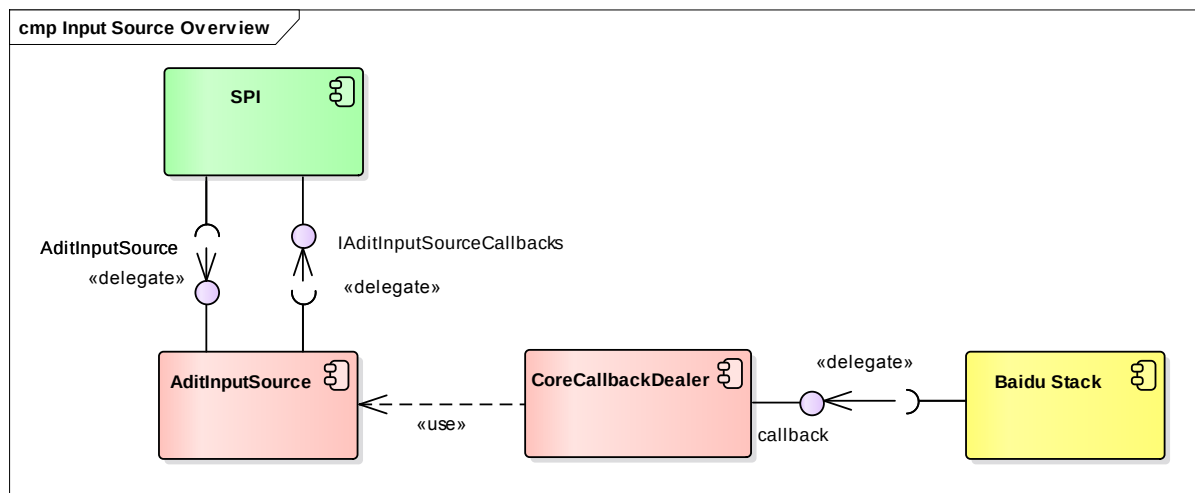


Figure 28 - Input Source Adaptation

The architecture and interaction with other classes is similar to other endpoints such as Video Sink or Audio Source.

AditInputSource class is provided by ADIT. The purpose of **AditInputSource** is to avoid interface changes in future, in case Input Source will be implemented using another technology than Wayland.

IAditInputSourceCallbacks is an interface provided by ADIT. Its methods are used to inform MC application, so that MC application can react accordingly.

6.6.1 USPI Input Source

The Input Source endpoint was extended by using the USPI Touch Backend library. USPI Touch Backend library simplifies the use of underlying platform components like Wayland. The interface of the USPI Touch Backend library is unified and independent from the underlying platform.

Figure 29 shows the structure of the USPI Input Source endpoint. The purpose, functionality and the interface to MC application is similar to the Wayland Input Source endpoint.

USPI Input Source uses shared wl-display. In this case, the wl-display is being shared by Video Sink endpoint and Input Source endpoint. The wl-display is being owned (created and destroyed) in context of the Video Sink endpoint and is used by Input Source endpoint.

The Touch callbacks are registered with the wl-display. The handler will invoke the respective callback when a specific touch input is given by the end user.

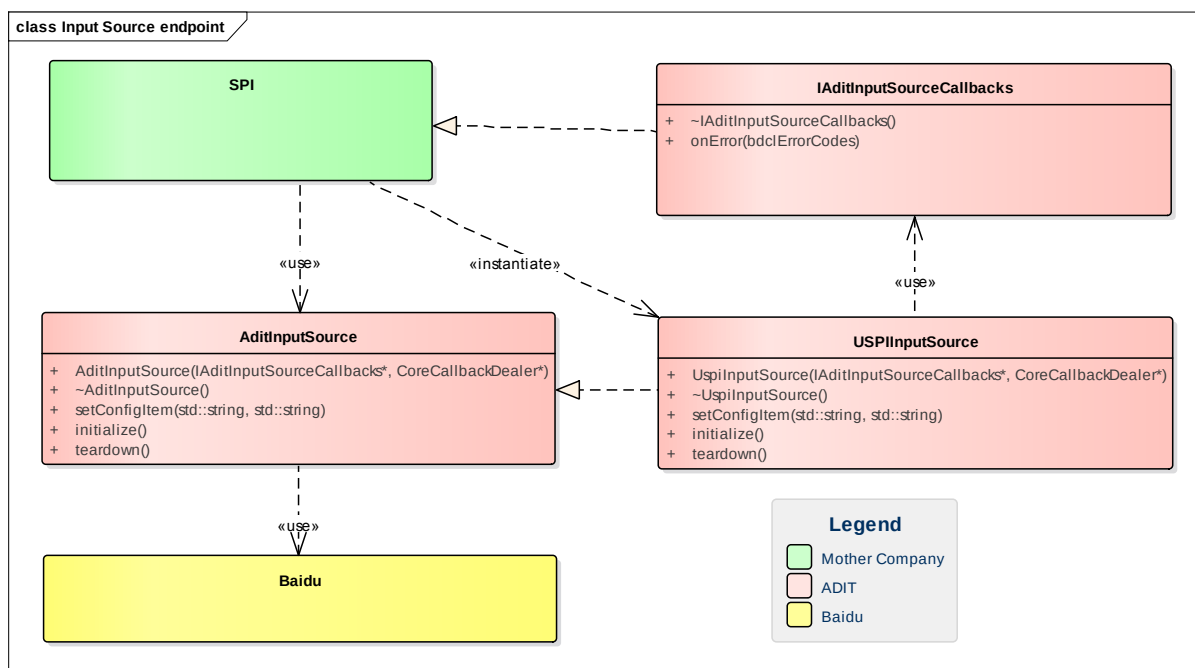


Figure 29 - USPI Input Source Endpoint

6.6.2 Input Source Initialization

Figure 30 shows the initialization of USPI Input Source endpoint.

MC application creates an object of class AditInputSource and setting configuration is not required in case of USPI, as we do not need any scaling by USPI touch for the touch points therefore we set 1 for the display size. This means that the touch points are divided by "1" and we get the touch points in display coordinates. After AditInputSource is initialized, MC application sets resolution of touch area to 1. Input Source Endpoint receives onTouch callback with touch event informations.

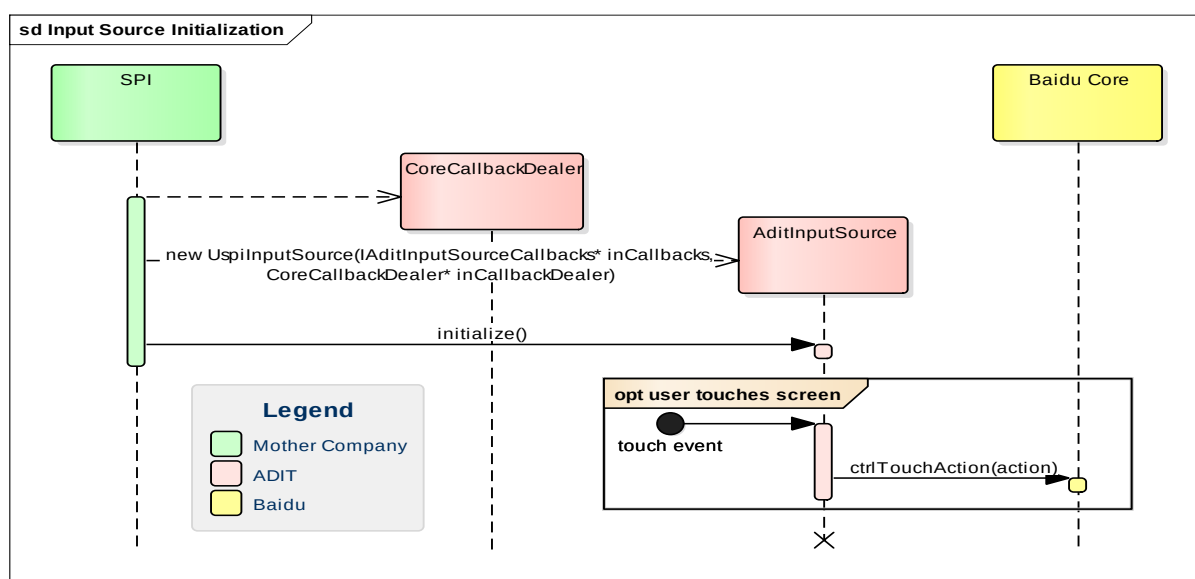


Figure 30 - Input Source Initialization

6.6.3 Input Source Touch Input Thread

Figure 31 shows creation, destruction, and touch handling of USPI Input Source. USPI Input Source uses the shared wl-display. Video and touch work on the same surface. The surface is created when video streaming starts and Video Sink endpoint shares the wl-display to USPI Touch Backend. USPI Input Source itself has no thread. The Input thread is created by USPI Wayland Touch when Video Sink endpoint shares the wl-display. USPI Touch Backend notifies USPI Input Source about touch and USPI Input Source reports the touch event to Baidu Core.

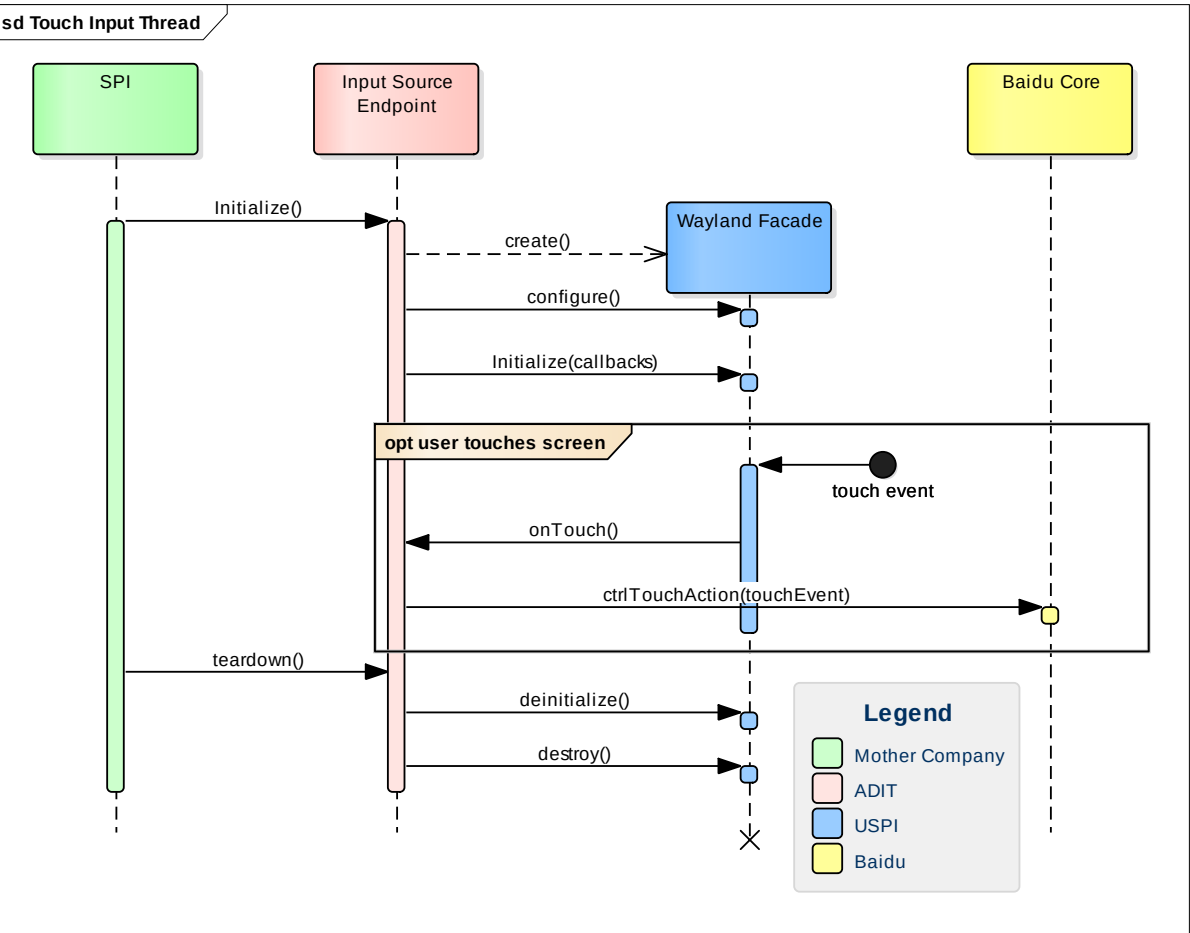


Figure 31 - Touch Input Thread

6.7 Debugging

6.7.1 DLT

BDCL uses DLT as main logging mechanism. MC application is responsible to create and register a DLT Application ID. MC application is responsible for declaring and registering APP with DLT.

BDCL declare and register the DLT Context.

void bdclRegisterCtxtWithDLT(void)

Description:

The API registers the DLT context of following packages

- bdcl_transport
- bdcl_init
- bdcl_core
- bdcl_audio
- bdcl_gst
- bdcl_input

void bdclDeRegisterCtxtWithDLT(void)**Description:**

The API deregisters the DLT context of above packages.

6.7.2 DLT Context

Name	Log Level	Description
BCRE	Fatal to Debug	Trace messages from the Baidu Core.
BTRA	Fatal to Verbose	BDCL Transport Trace messages from AOAP Transport during an wired BDCL session over USB
BINT	Fatal to Verbose	ADIT Wrapper to the Baidu Core This also logs the messages from Baidu core.
BAUD	Fatal to Verbose	BDCL Audio endpoint Trace messages of package adit-bdcl-audio which includes the AudioSink endpoint and AudioSource endpoint. Log messages including the logging of the communication to the audio architecture (e.g. ALSA) if high log level (debug, verbose) was enabled.
BVID	Fatal to Verbose	BDCL Video endpoint Trace messages of package adit-bdcl-video which includes the GStreamer specific video implementation.
BINP	Fatal to Verbose	BDCL Input endpoint Trace messages of package adit-bdcl-input and provides information about touch events.

Table 5 - DLT Context Description

6.8 Channel ID

It is a unique Head Unit identification code to authenticate the Head Unit by Baidu app in the MD. A valid Channel ID is required to establish the successfully Baidu Session. Head Unit has to send a valid Channel ID in the beginning of the session through MSG_CMD_STATISTIC_INFO message. These Channel ID's are project specific and distributed by Baidu and features of Baidu CarLife are customized based on the Channel ID.

MC has to set this value through setCarlifeStatisticInfo API. For further details, please refer Interfaces for BDCL Platform

6.9 Deviation in the BDCL app behavior against Baidu specification

Some of the features of Baidu app behaves differently against mentioned in the specification by Baidu and it behaves differently for different types of devices (Android and iOS). Since Baidu confirmed that these kind of behavior will not be changed ADIT has provided workaround to deal with these kind of use cases. Some of the examples given below.

6.9.1 Use case: pause and play Media Audio

Baidu app behaves differently for the use case of Media Audio pause and play for different devices is explained through below sequence diagram.

6.9.1.1 iOS Device:

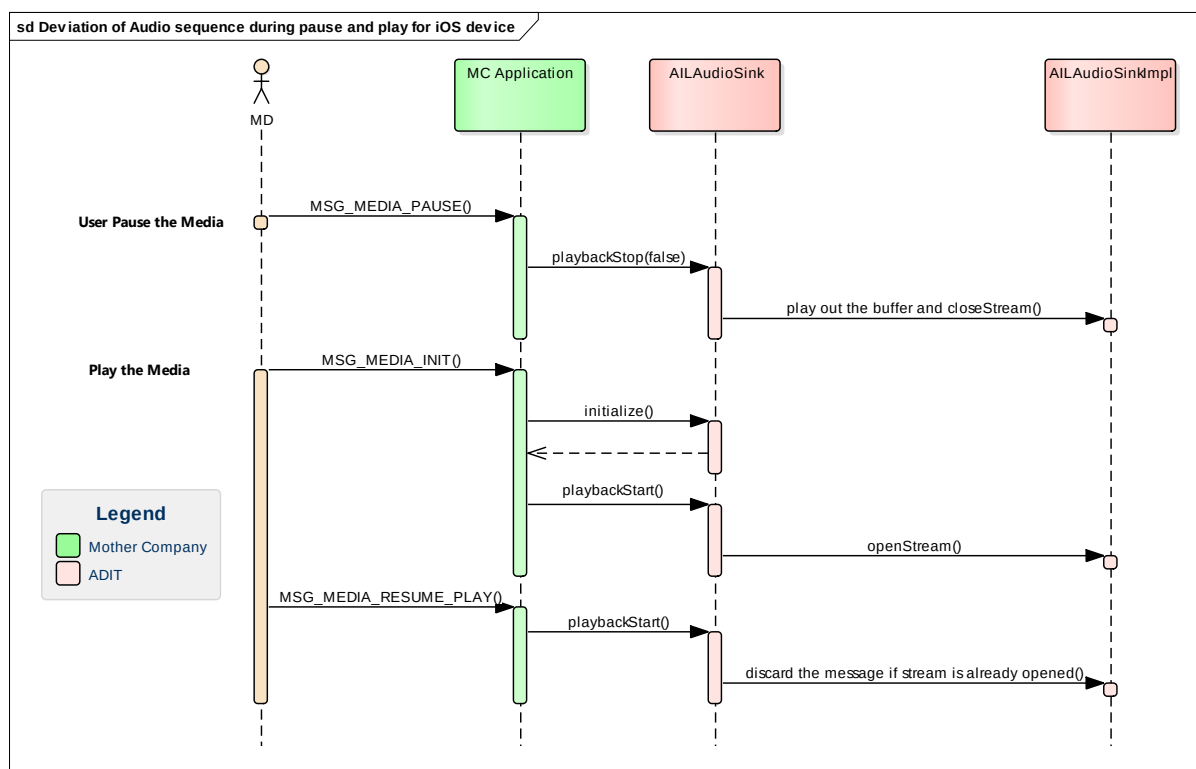


Figure 32 - iOS device behavior for pause and play Media Audio

6.9.1.2 Android Device:

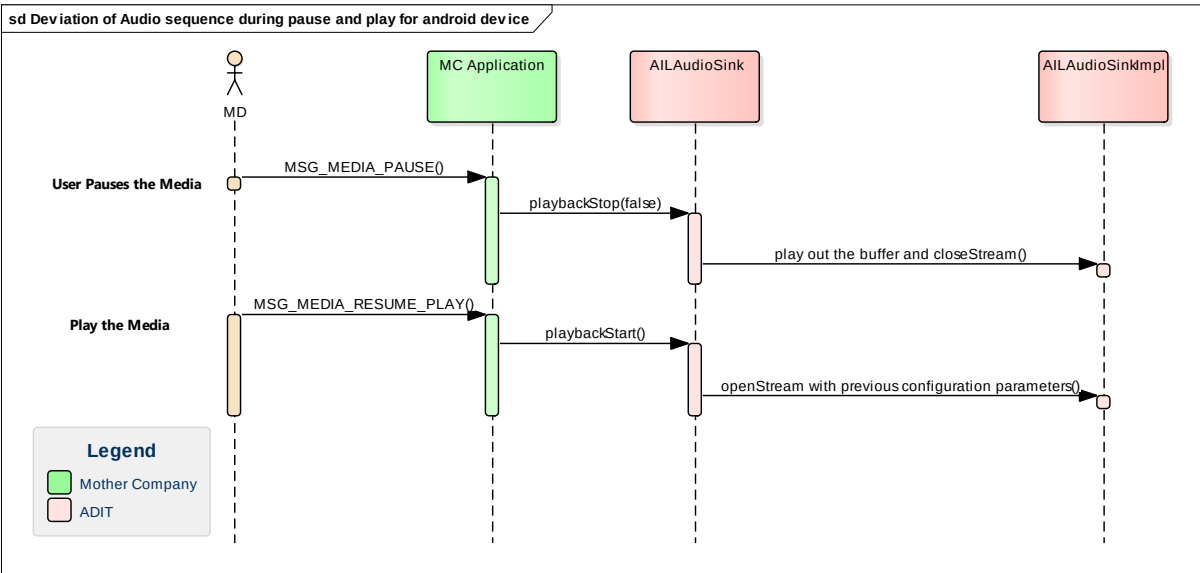


Figure 33 - Android device behavior for pause and play Media Audio

6.9.2 Use case: Media Audio track change

Baidu app behaves differently for the use case of Media Audio track change for different devices is explained through below sequence diagram.

6.9.2.1 iOS Device:

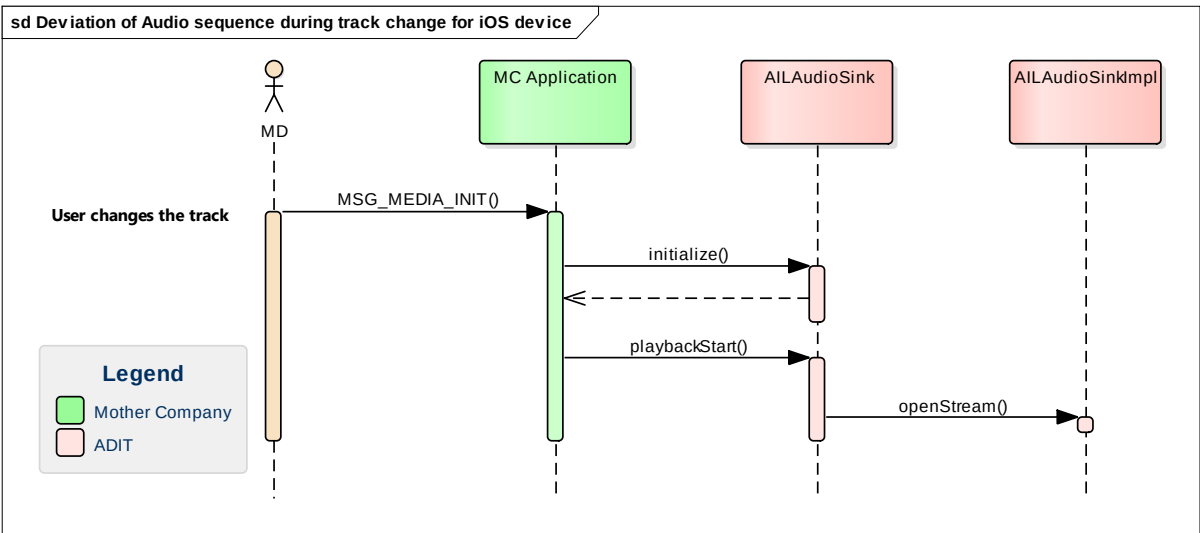


Figure 34 - iOS device behavior for Media Audio track change

6.9.2.2 Android Device:

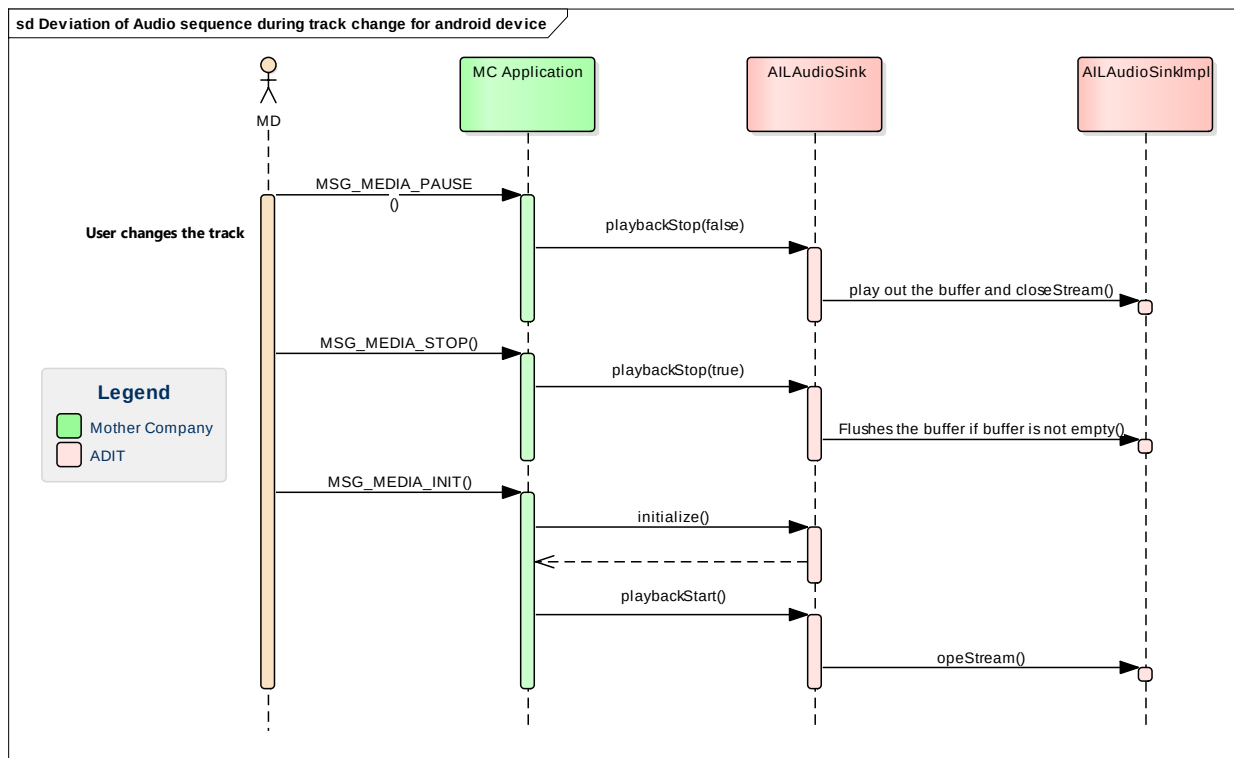


Figure 35 - Android device behavior for Media Audio track change

6.9.3 Audio Sample rate of 48 kHz for iOS device.

Default sample rate for media audio is 44.1 kHz but it's mentioned in the Baidu Specification that we can configure the Baidu app to stream media audio at 48 kHz.

This works as expected for Android device but its observed that iOS device will stream media audio data at 44.1 kHz in spite of configuring the MD to stream at 48 kHz.

It's also confirmed by Baidu that iOS device continue to behave in the same manner.