Android RACE

Low Level Design

Media Player

AllGo Embedded Systems Pvt Ltd

Version History

Version Number	Date	Author	Remarks
0.1	19 June 2019	Kaustubh Ginde	Initial Draft
0.2	27 June 2019	Kaustubh Ginde	Updated based on review comments
0.3	28 June 2019	Kaustubh Ginde	Updated based on comments
1.0	3 July 2019	Kaustubh Ginde	Baselined

Table of Contents

1 Intodunction....3
  1.1 Purpose....3
  1.2 Overview....3
  1.3 References....3
2 Low-Level Architecture....4
  2.1 Media Player Client....6
  2.2 Media Player Server....6
  2.3 MTP Player and JNI....6
  2.4 IAP Player....7
3 Sequence Diagram....8
  3.1 Media Player Client – Server Communication....8
  3.2 MTP Seek Sequence....9
  3.3 MTP Data Source Flow....10
  3.4 IAP Double Buffer Operation....10
  3.5 IAP Double buffer control flow....11
  3.6 IAP Playback thread operation....13
4 Class Design....14
  4.1 Class Diagram....14
  4.2 Sequence flow diagram....15
  4.3 Collaboration Diagram....16
5 File Structure....17
6 Interfaces....17

1 Intoduction

1.1 Purpose

This document describes the low-level design of Media Player from Android RACE. Media Player is responsible for implementing the player functionalities provided by RACE in the Android environment.

1.2 Overview

Android Media Player is designed to implement the RACE player functionalities in Android. This module serves to provide a client-server mechanism between RACE and Android layer for performing player functionalities on MTP and IAP devices.

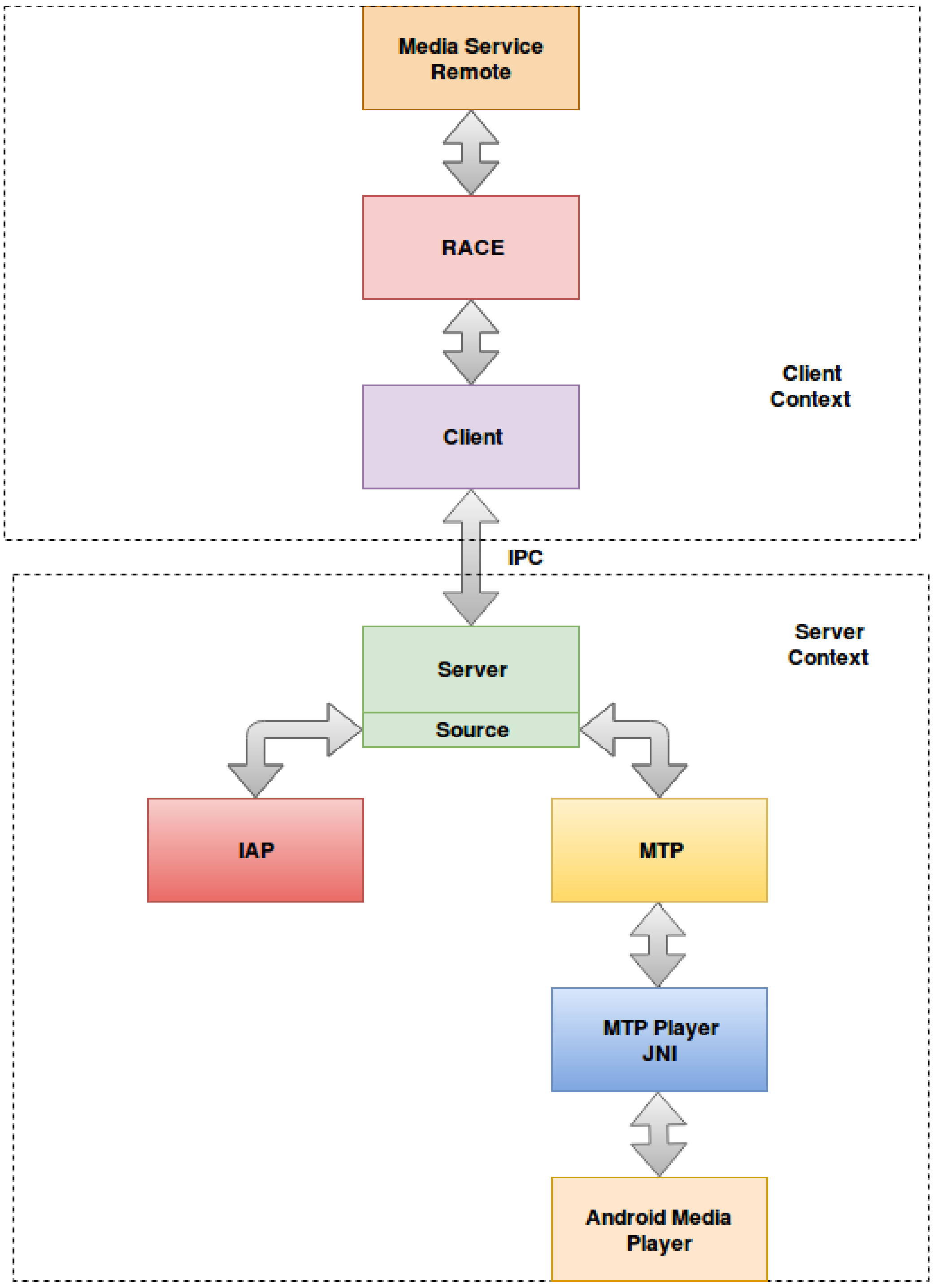
Media Player mainly comprises of a native layer and a Java layer. The native layer consists of the Media Player Client and Server along with a device type based implementation that is categorized into IAP and MTP based functionalities. The Java layer allows the MTP Media Player to utilize the Android player functionalities. Each component is described in detail in further sections.

1.3 References

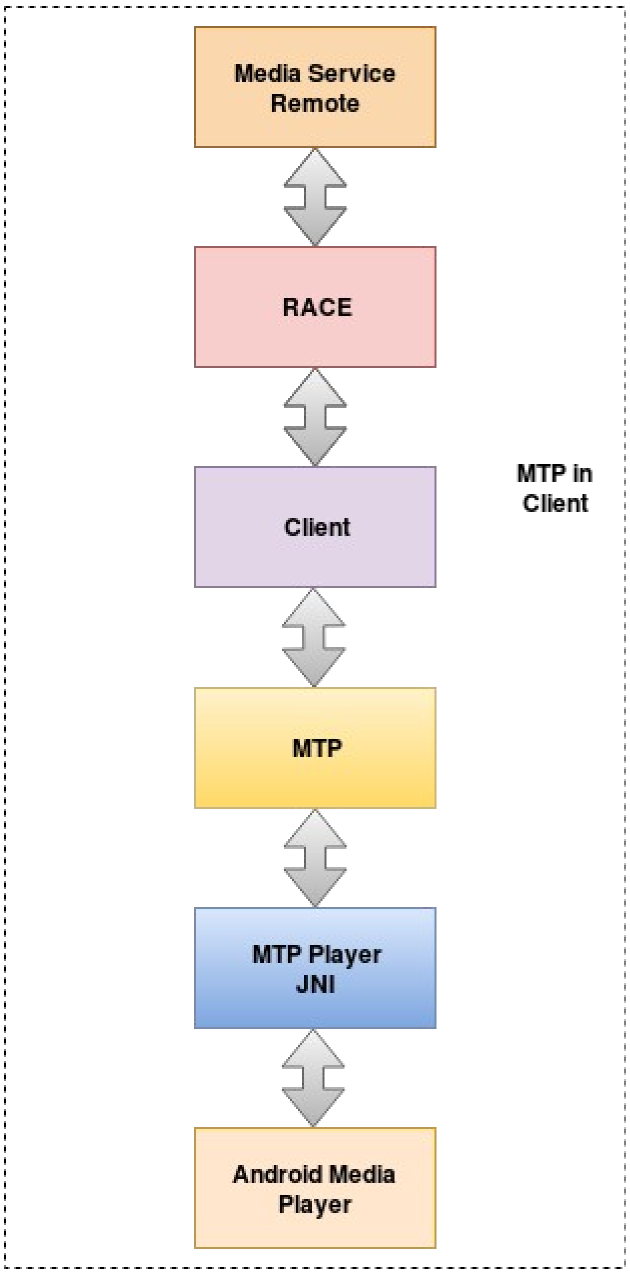
- Media_Service_Remote_LLD_v1.0.doc

2 Low-Level Architecture

Android Media Player is designed to achieve player functionalities exposed by RACE SDK to Media Service Remote. The different operational layers are as shown below :



Android RACE Media Player
(Client – Server approach)



MTP In Client Approach

MTP playback can be achieved with two approaches, MTP in Client and Client-Server approach. The first approach includes using MTP in client context. In this approach, all the MTP functions will be invoked from media player client and there is no dependency on the server. The client-server approach relies on the use of two threads, one EventThread in the server and one on the client side. These two threads use three message queues for communication. One message queue for sending command messages from client to server, one for sending the response from the server to the client and a third queue for sending callbacks to the callback thread from server to client. The two approaches are depicted in the diagrams above. The detailed usage is explained in further sections.

2.1 Media Player Client

Media Player client provides API implementations of android media player client side for Media Service Remote.

The client can be utilized in a dual way in case of MTP playback. In the first approach, the MTP playback functions discussed in the following section can be invoked directly from the client. This is provided as a compile-time macro “MTP_IN_CLIENT”. As depicted in the second diagram above, in the case of MTP in Client all the MTP functionalities will be invoked directly by the client and there is no interaction with the server.

In the second approach, the client can be used in a regular client-server logic. In the client-server approach, the client is responsible for creating an EventThread for listening to callbacks from the media player server and thereby invoke the callbacks registered for specific events. Callback can be registered using race_mp_register_callback() function. The client creates a message for each player functionality that is requested and sends this message to the server for execution. For sending this message it uses the MediaPlayer send message queue. For every message sent over this send MQ, it will receive a response from the server on the MediaPlayer receive message queue. The client also maintains a handle in the form of MPClientHandle structure which consists of all the necessary information required to control playback.

IAP playback functionalities are provided using the client-server approach for android media player.

The different interfaces exposed by the media player client are detailed in the interface section of this document.

2.2 Media Player Server

Media player server is responsible for processing the message sent by the client. For this, the server creates an EventThread for processing the command messages received from media player client. It also creates three message queues which are MediaPlayer send and receive message queues for receiving command messages from the client and sending response respectively and an event message queue to send callback related messages to the client.

The server will process any commands that it receives and invokes the source functions, which are responsible for differentiating between IAP and MTP and invoke the respective asynchronous function for that specific capability. For example, when server “race_mp_open()” is called, source will invoke race_mp_iap_open() or race_mp_mtp_open based on the eDeviceType parameter. This will be the approach for all the other functions invoked by the server.

2.3 MTP Player and JNI

MTP player handles all the MTP playback operations. For this, it makes use of the Android Media Player. MTP player uses JNI to access MTPMediaPlayer, MTPMetadataRetriever and MTPDataSource JAVA classes, to achieve the playback functionalities and invoke the Android

Media Player APIs.

The JNI is a native programming interface. It allows Java code that runs inside a Java Virtual Machine (VM) to interoperate with applications and libraries written in other programming languages, such as C, C++, and assembly.

- The MTPMediaPlayer class invokes the Android media player APIs like play(), pause(), etc, when invoked by the native mtp_player. JNI allows the native layer to instantiate and use the methods exposed by MTPMediaPlayer class. MTPMediaPlayer also invokes callbacks like jniOnSeekComplete() when onSeekComplete() is triggered by Android media player.
- MTPMetaDataRetriever class is mainly used by the native layer for accessing the metadata information by using Android internal classes like BitmapFactory. This class exposes methods like getAlbumArt(), extractMetadata() to the native layer using JNI.
- MTPDataSource is used for invoking native functions to retrieve data from devicemanager. For example, when MediaDataSource (android internal class) triggers readAt() function, the native function jniReadAt() is invoked using JNI. This internally invokes the readAt() method defined as a part of mtp_data_source.

MTP Playback Sequence diagram in the next section provides a detailed view of this interaction.

2.4 IAP Player

IAP player implements the playback operations with the use of three threads namely, iap_write_thread, iap_fill_buffer_thread, iap_playback_thread. IAP player directly interacts with the device pcm node for providing audio playback. When initialized, IAP player allocates an IAPHandle structure which maintains the necessary information like, audio frame count, pcm handles and IAP specifications needed by ALSA for reading the data. IAP player uses TinyALSA for handling audio.

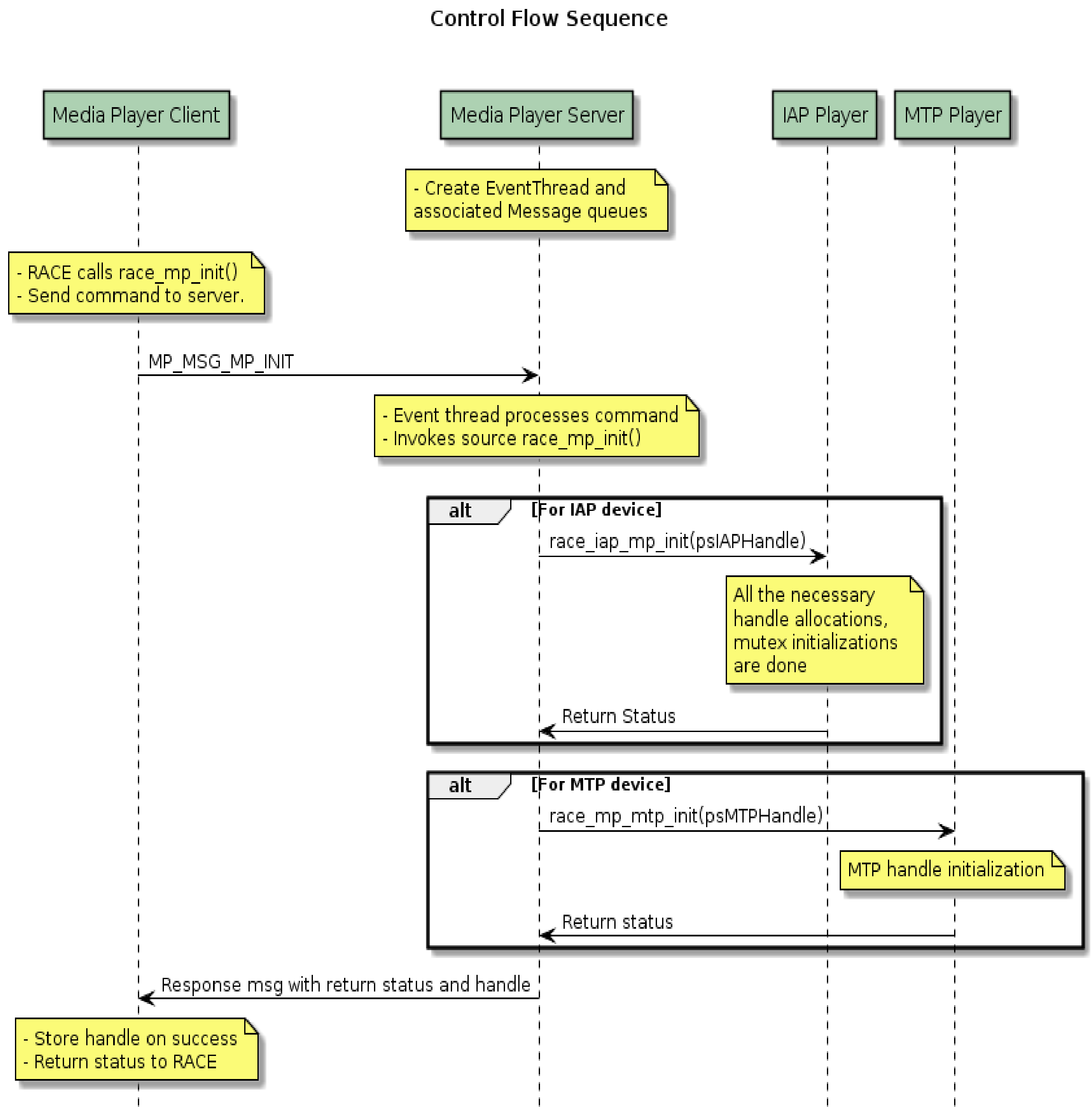
In the case of IAP two approaches are possible to achieve pcm data read/write. In the first approach called Audio double buffer approach, two threads mp_race_iap_fill_buffer_thread and mp_race_iap_write_buffer_thread are used. The mp_race_iap_fill_buffer_thread will read the data from the input PCM node into a buffer and signal the mp_race_iap_write_buffer_thread to read the data from that buffer. The mp_race_iap_write_buffer_thread will pick the data from that buffer and write it to audio track using ALSA API for audio playback.

The second approach involves the use of a single thread, race_iap_playback_thread. This thread handles two basic operations. This thread reads audio data from the PCM input node into a buffer and writes this data either to the AudioTrack buffer using ALSA API or writes the data from the buffer directly to the audio output device's PCM node. IAP playback sequence diagram shows this interaction.

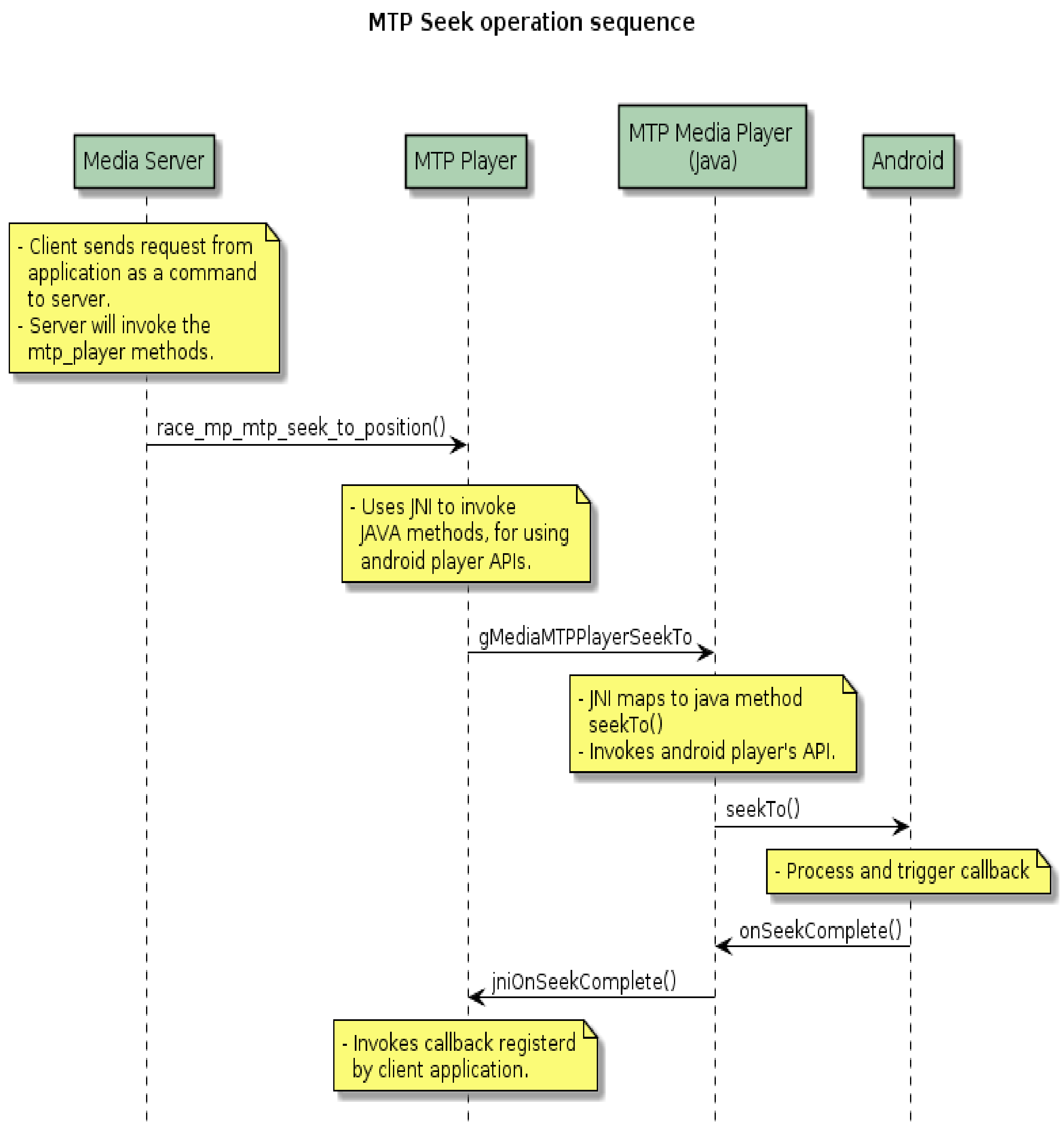
3 Sequence Diagram

The following sequence diagrams depict the flow of control in Android RACE media player.

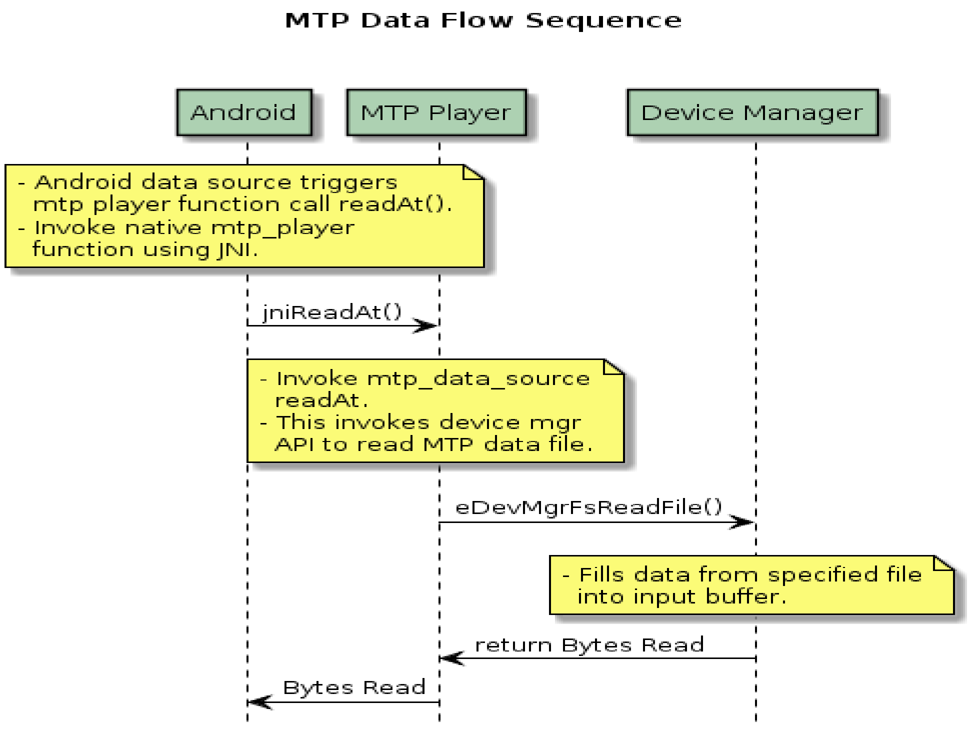
3.1 Media Player Client – Server Communication


Control Flow Sequence
Media Player Client
Media Player Server
IAP Player
MTP Player
- Create EventThread and associated Message queues
- RACE calls race_mp_init()
- Send command to server.
MP_MSG_MP_INIT
- Event thread processes command
- Invokes source race_mp_init()
alt
[For IAP device]
race_iap_mp_init(psIAPHandle)
All the necessary handle allocations, mutex initializations are done
Return Status
alt
[For MTP device]
race_mp_mtp_init(psMTPHandle)
MTP handle initialization
Return status
Response msg with return status and handle
- Store handle on success
- Return status to RACE

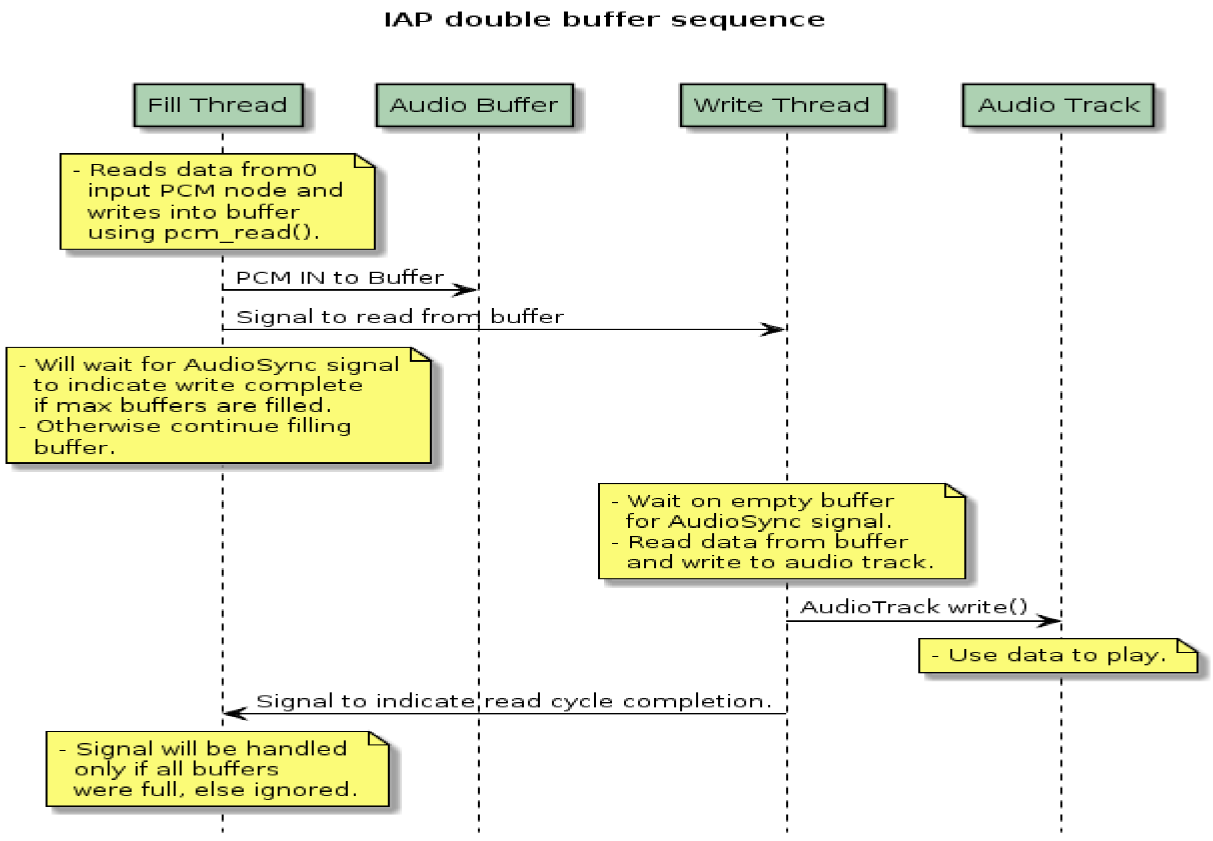
3.2 MTP Seek Sequence


MTP Seek operation sequence
Media Server
MTP Player
MTP Media Player (Java)
Android
- Client sends request from application as a command to server.
- Server will invoke the mtp_player methods.
race_mp_mtp_seek_to_position()
- Uses JNI to invoke JAVA methods, for using android player APIs.
gMediaMTPPlayerSeekTo
- JNI maps to java method seekTo()
- Invokes android player's API.
seekTo()
- Process and trigger callback
onSeekComplete()
jniOnSeekComplete()
- Invokes callback registerd by client application.

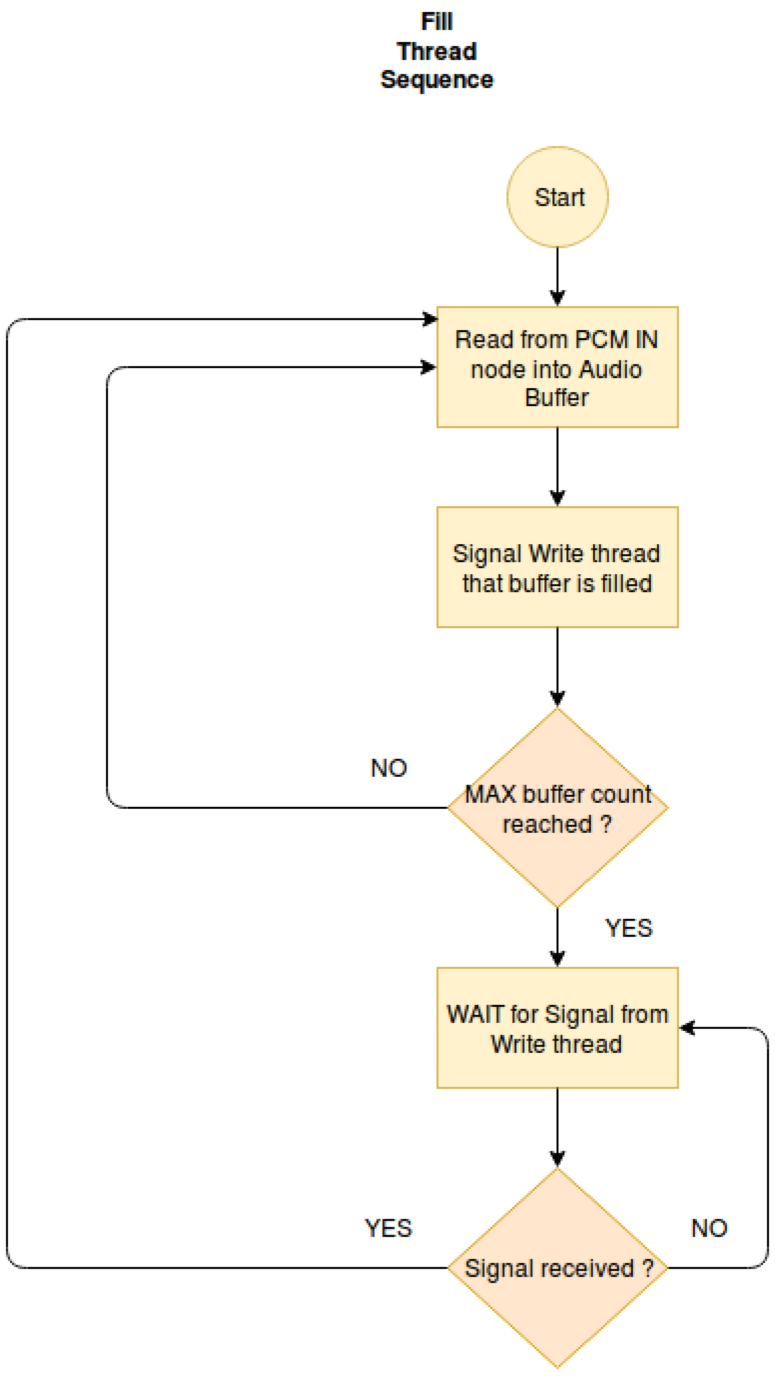
3.3 MTP Data Source Flow


MTP Data Flow Sequence
Android
MTP Player
Device Manager
- Android data source triggers mtp player function call readAt().
- Invoke native mtp_player function using JNI.
jniReadAt()
- Invoke mtp_data_source readAt.
- This invokes device mgr API to read MTP data file.
eDevMgrFsReadFile()
- Fills data from specified file into input buffer.
return Bytes Read
Bytes Read

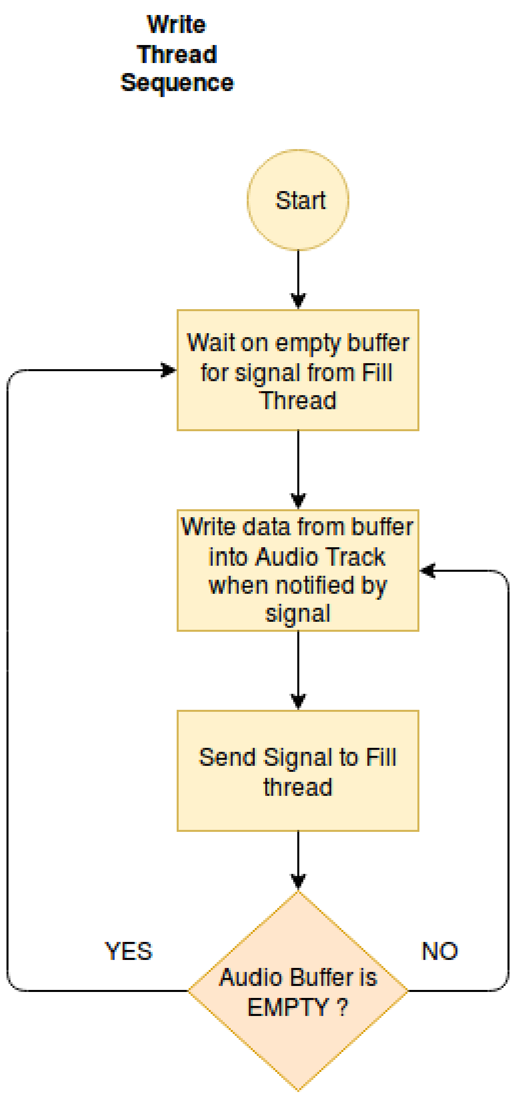
3.4 IAP Double Buffer Operation


IAP double buffer sequence
Fill Thread
Audio Buffer
Write Thread
Audio Track
- Reads data from0 input PCM node and writes into buffer using pcm_read().
PCM IN to Buffer
Signal to read from buffer
- Will wait for AudioSync signal to indicate write complete if max buffers are filled.
- Otherwise continue filling buffer.
- Wait on empty buffer for AudioSync signal.
- Read data from buffer and write to audio track.
AudioTrack write()
- Use data to play.
Signal to indicate read cycle completion.
- Signal will be handled only if all buffers were full, else ignored.

3.5 IAP Double buffer control flow


Fill Thread Sequence
Start
Read from PCM IN node into Audio Buffer
Signal Write thread that buffer is filled
NO
MAX buffer count reached ?
YES
WAIT for Signal from Write thread
YES
NO
Signal received ?

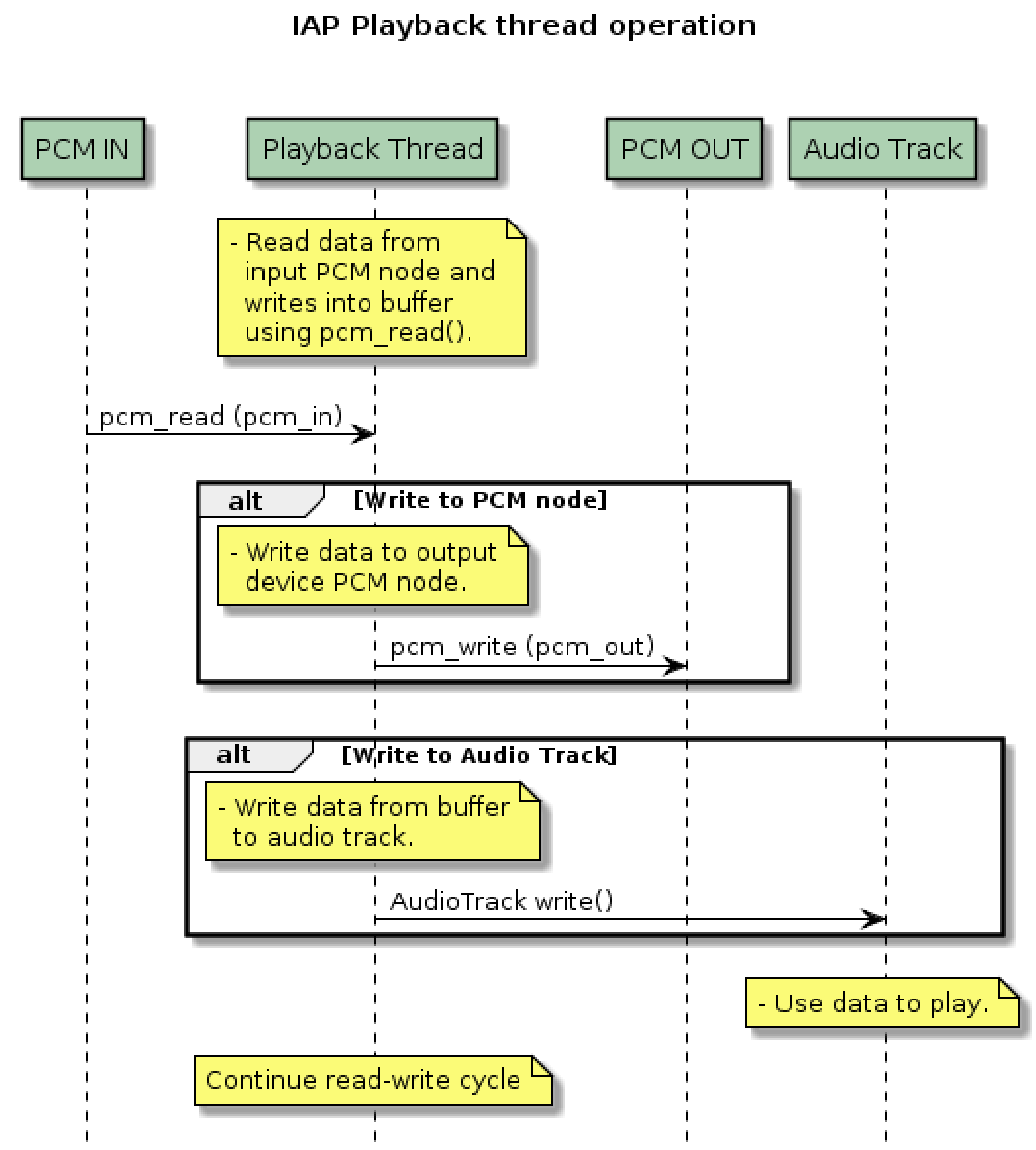
IAP Double Buffer – Fill Thread Flow


Write Thread Sequence
Start
Wait on empty buffer for signal from Fill Thread
Write data from buffer into Audio Track when notified by signal
Send Signal to Fill thread
Audio Buffer is EMPTY ?
YES
NO

IAP Double Buffer- Write Thread Flow

The two threads described in the flowcharts above will run parallelly. AudioSync signal helps to achieve the synchronization in read and write operations based on the buffer fill status.

3.6 IAP Playback thread operation

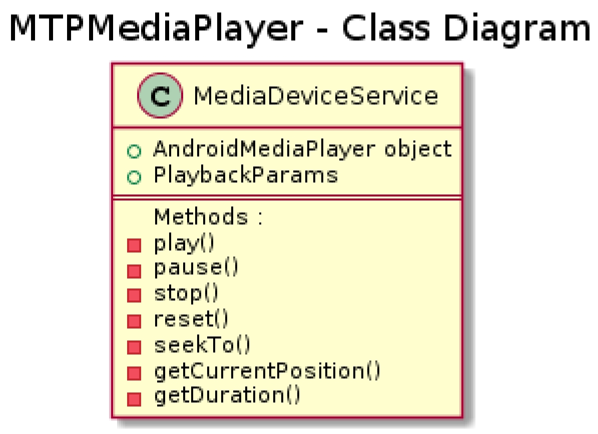


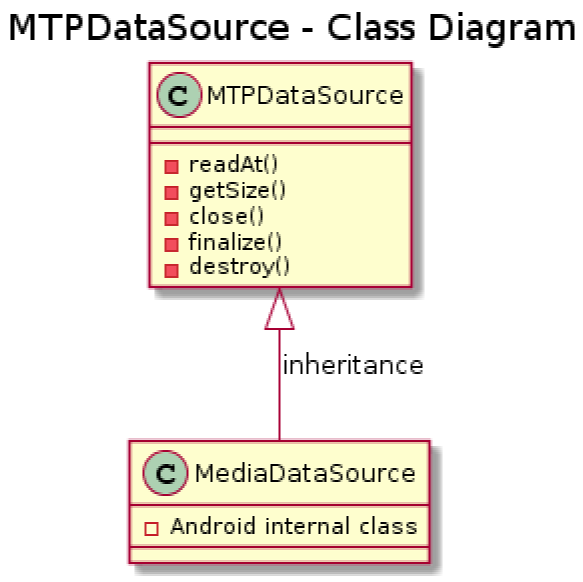
IAP Playback thread operation
PCM IN
Playback Thread
PCM OUT
Audio Track
- Read data from input PCM node and writes into buffer using pcm_read().
pcm_read (pcm_in)
alt [Write to PCM node]
- Write data to output device PCM node.
pcm_write (pcm_out)
alt [Write to Audio Track]
- Write data from buffer to audio track.
AudioTrack write()
- Use data to play.
Continue read-write cycle

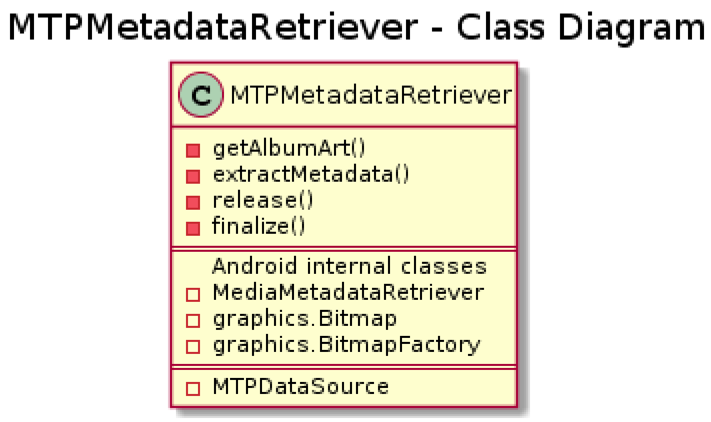
4 Class Design

4.1 Class Diagram

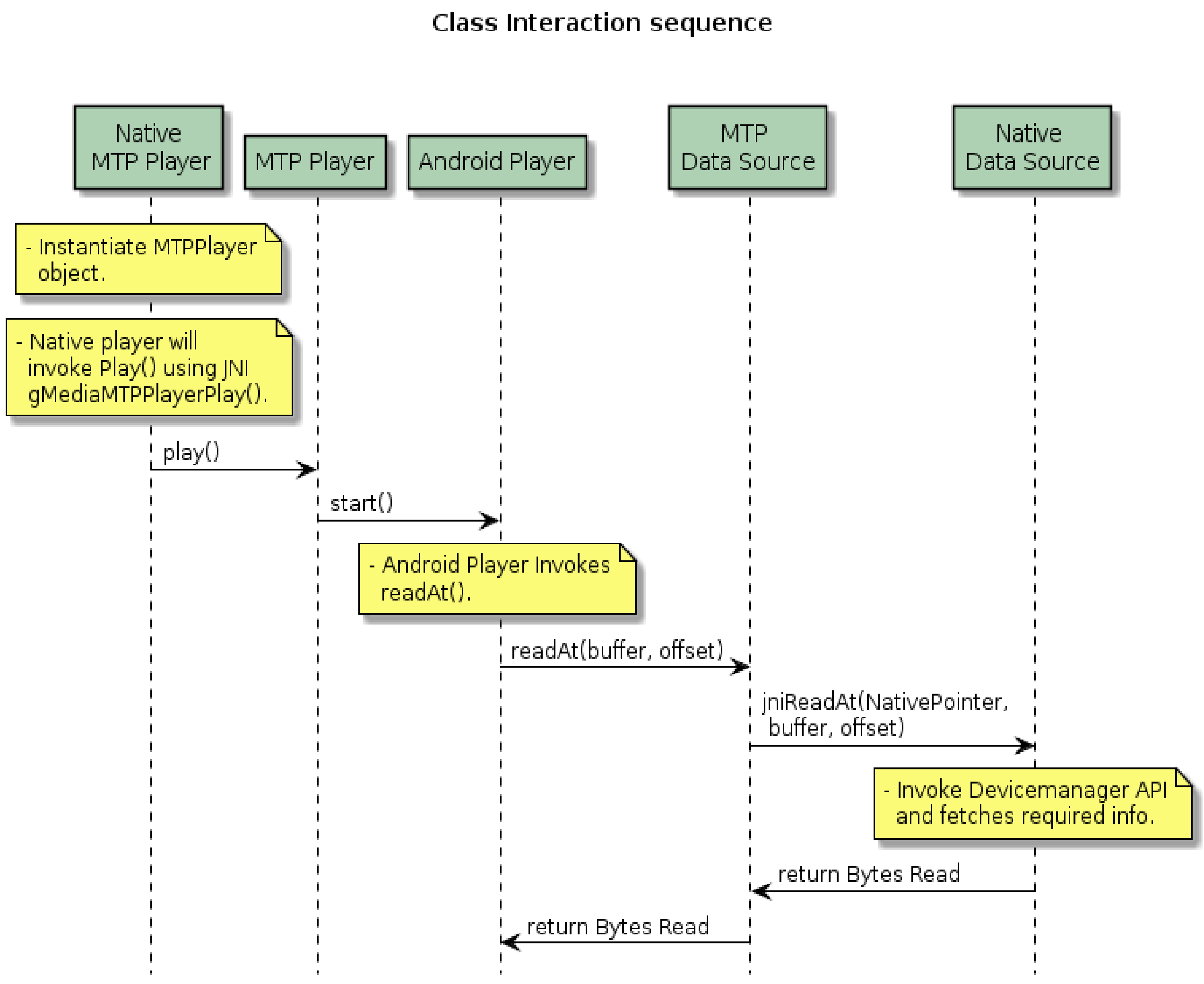
The MTPMediaPlayer, MTPMetadataRetriever and MTPDataSource JAVA classes are as depicted below.







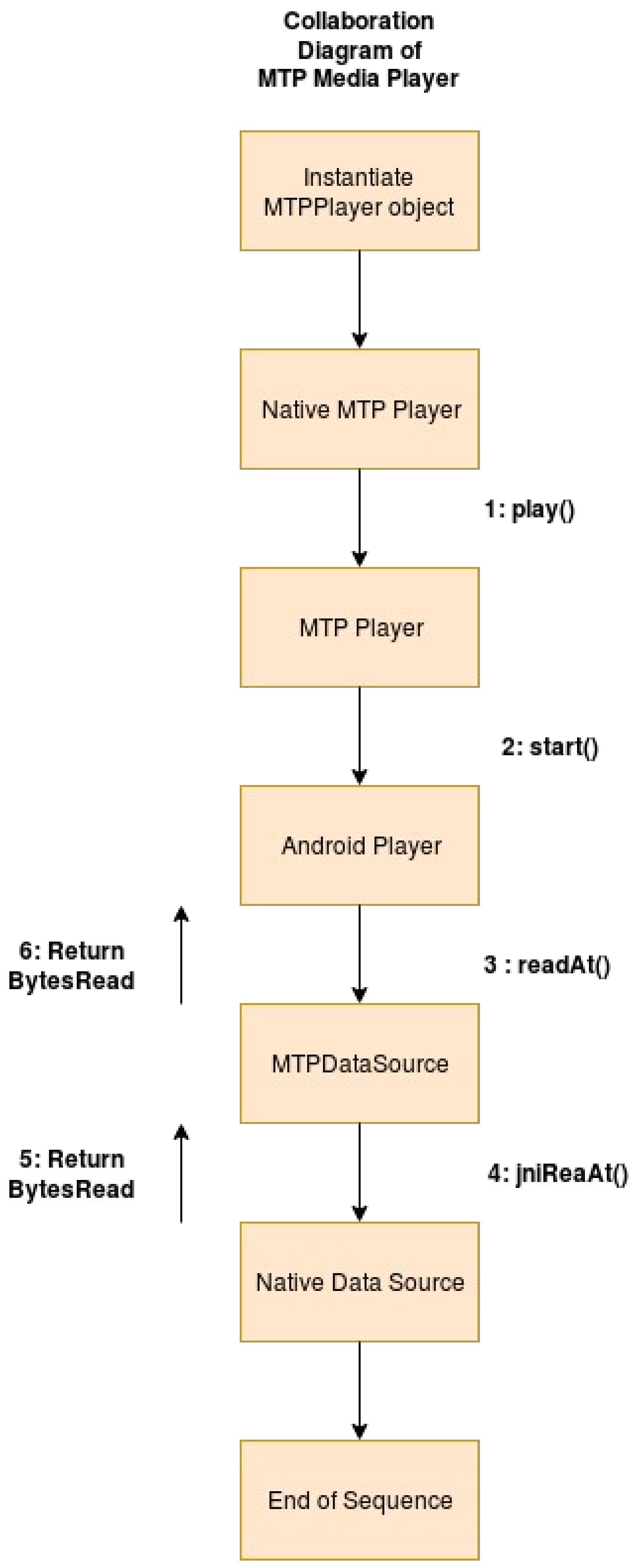
4.2 Sequence flow diagram


Class Interaction sequence
Native MTP Player
MTP Player
Android Player
MTP Data Source
Native Data Source
- Instantiate MTPPlayer object.
- Native player will invoke Play() using JNI gMediaMTPPlayerPlay().
play()
start()
- Android Player Invokes readAt().
readAt(buffer, offset)
jniReadAt(NativePointer, buffer, offset)
- Invoke Devicemanager API and fetches required info.
return Bytes Read
return Bytes Read

Native MTP Player is responsible for instantiating MTPPlayer and MTPDataRetriever java classes. MTPDataRetriever will internally set the MTPDataSource (java) reference for Android.

In the above diagram MTP Player, Android, Data source and native data source represent class objects and the sequence diagram depicts the methods invoked from corresponding classes.

4.3 Collaboration Diagram



5 File Structure

- The client end logic for android race media player is defined in *android_race_media_player_client.cpp* .
- The logic for android race media player server is defined in *android_race_media_player_server.cpp.*
- Media player server source is contained in *android_race_media_player.cpp*
- IAP player functionalities are implemented in iap_player.cpp.
- mtp_player.cpp consists of the MTP player JNI calls. The JNI implementations for Java methods is in MTPMediaPlayer.java .
- mtp_data_source.cpp implements the methods to interact with device manager. MTPDataSource.java forms the corresponding JAVA layer.

The JAVA files for MTP player implementation are MTPMediaPlayer.java, MTPMetadataRetriever.java, and MTPDataSource.java. MTPMediaPlayer.java has the JNI implementations of various player functionalities. MTPMetadataRetriever.java is used for fetching data like album-art.

6 Interfaces

Android race media player client exposes the following interfaces :

- race_mp_init()
- race_mp_deinit()
- race_mp_check_cleanup ()
- race_mp_open()
- race_mp_register_callback()
- race_mp_seek_to_position()
- race_mp_get_stream_duration()
- race_mp_get_stream_position_and_lba()
- race_mp_get_playing_state()
- race_mp_set_playing_state()

- race_mp_get_metadata()
- race_mp_set_iap_param()
- race_mp_get_play_track_info()
- race_mp_get_albumart_info()

Following functions are implemented by **iap_player** :

- race_iap_mp_init()
- race_mp_iap_deinit()
- race_mp_iap_open()
- race_mp_iap_check_cleanup()
- race_mp_iap_get_playing_state()
- race_mp_iap_set_playing_state()
- race_mp_iap_set_param()

Following functions are implemented by **mtp_player** :

- jniInitNativeMediaDataSource()
- jniDestroyNativeMediaDataSource()
- jniCloseMediaDataSource()
- jniGetSize()
- jniReadAt()
- jniOnSeekComplete()
- jniOnError()
- jniOnPlaybackComplete()
- race_mp_mtp_open()
- race_mp_mtp_set_playing_state()
- race_mp_mtp_check_cleanup()

- race_mp_mtp_seek_to_position()
- race_mp_mtp_get_metadata()
- race_mp_mtp_get_play_track_info()
- race_mp_mtp_get_stream_position_and_lba()
- race_mp_mtp_get_albumart_info()
- race_mp_mtp_get_stream_duration()
- race_mp_mtp_init()
- race_mp_mtp_register_callback()
- race_mp_mtp_get_playing_state()