Qualcomm Technologies, Inc.

Application to MSDC

Interface Specification for MSDC Release 4.4.00.00

80-62893-1 Rev. AA

April 6, 2023

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
AA	April 2023	Initial release

Contents

Figures

1 Introduction

1.1 Purpose

This document defines the interface specification that exists between the MSDC SDK and the application (app) on the user equipment (UE). This document is intended for users who are familiar with iOS and MacOS app development (including related concepts) and the media player interaction of the app.

These concepts are outside the scope of this document:

- Enhanced Multimedia Broadcast Multicast Services (eMBMS)
- Dynamic Adaptive Streaming over HTTP (DASH)

1.2 Environment

The following environments are required for a device to run an application with MSDC SDK.

Software requirements

- MacOS (12+)
- XCode (v14+)
- iOS (14+)
- DASH Player (for iOS and MacOS)

Hardware requirements

- MacBook Air (M1, 2020) and above
- iPhone 6S and above
- MSDC 4.4 support on Qualcomm CPE

1.3 Support

For support, visit the LTE Broadcast SDK web page on the Qualcomm® Developer Network (QDN): https://developer.qualcomm.com/ltebroadcast.

2 Overview

The Qualcomm MSDC enables efficient and high-quality delivery of live and non-real time media services over LTE eMBMS networks. The MSDC is a middleware platform that provides a core set of eMBMS services in addition to implementing MSDC SDK APIs for iOS and MacOS.

The MSDC SDK APIs allow applications to access eMBMS services without requiring detailed knowledge of eMBMS and the complexities involved with data delivery over a broadcast network.

The following figure shows the overall architecture of these interfaces on the iOS and MacOS device.

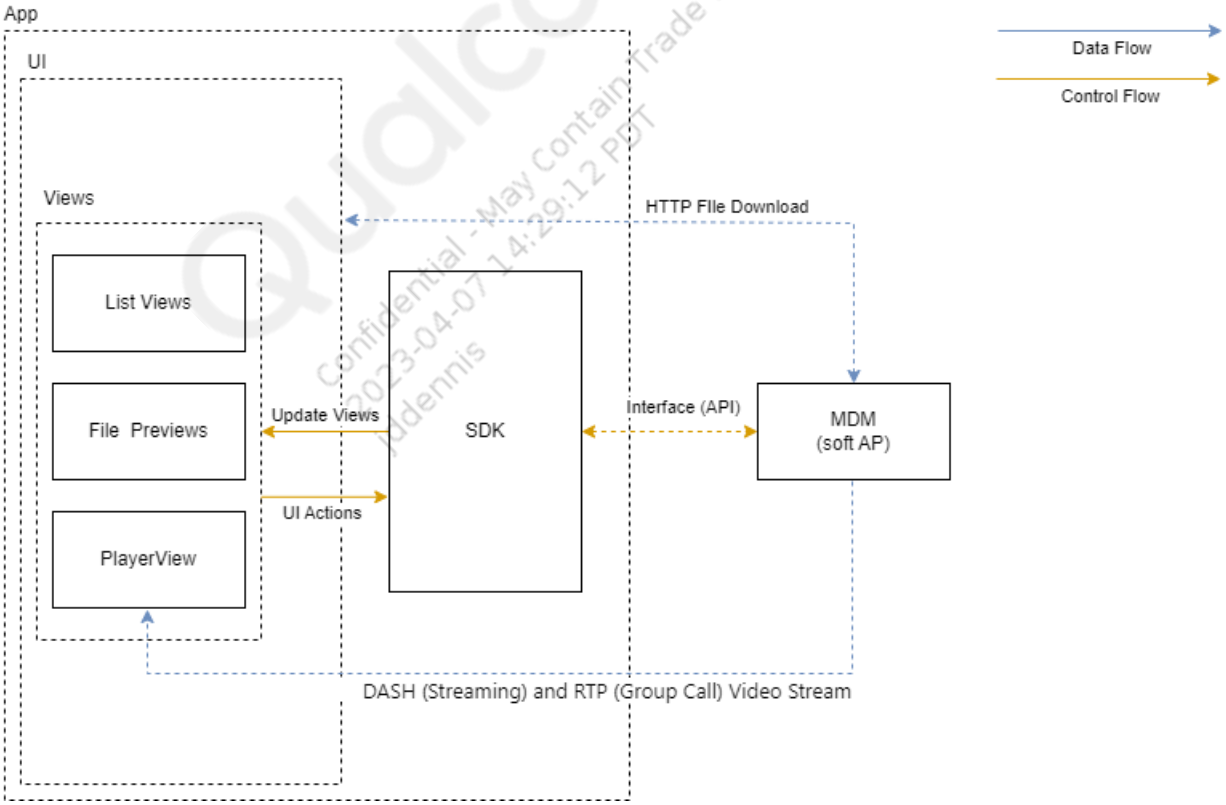


Figure 2-1 MSDC SDK and app interfaces

The MSDC API provides model and controller implementation for a typical MVC-based UI application. This architecture facilitates quick development of client applications.

The MSDC API allows developers to concentrate on creating apps for live streaming, group call and file delivery services, instead of worrying about the underlying iOS/MacOS Services and management of communication with those services. The MSDC.xcframework library (MSDC SDK) and sample UI source code for iOS/MacOS are available on QDN: https://developer.qualcomm.com/software/lte-broadcast-sdk.

3 Streaming service

This chapter describes the MSDC API descriptions and call flow sequences for an application that provides a Streaming service to users.

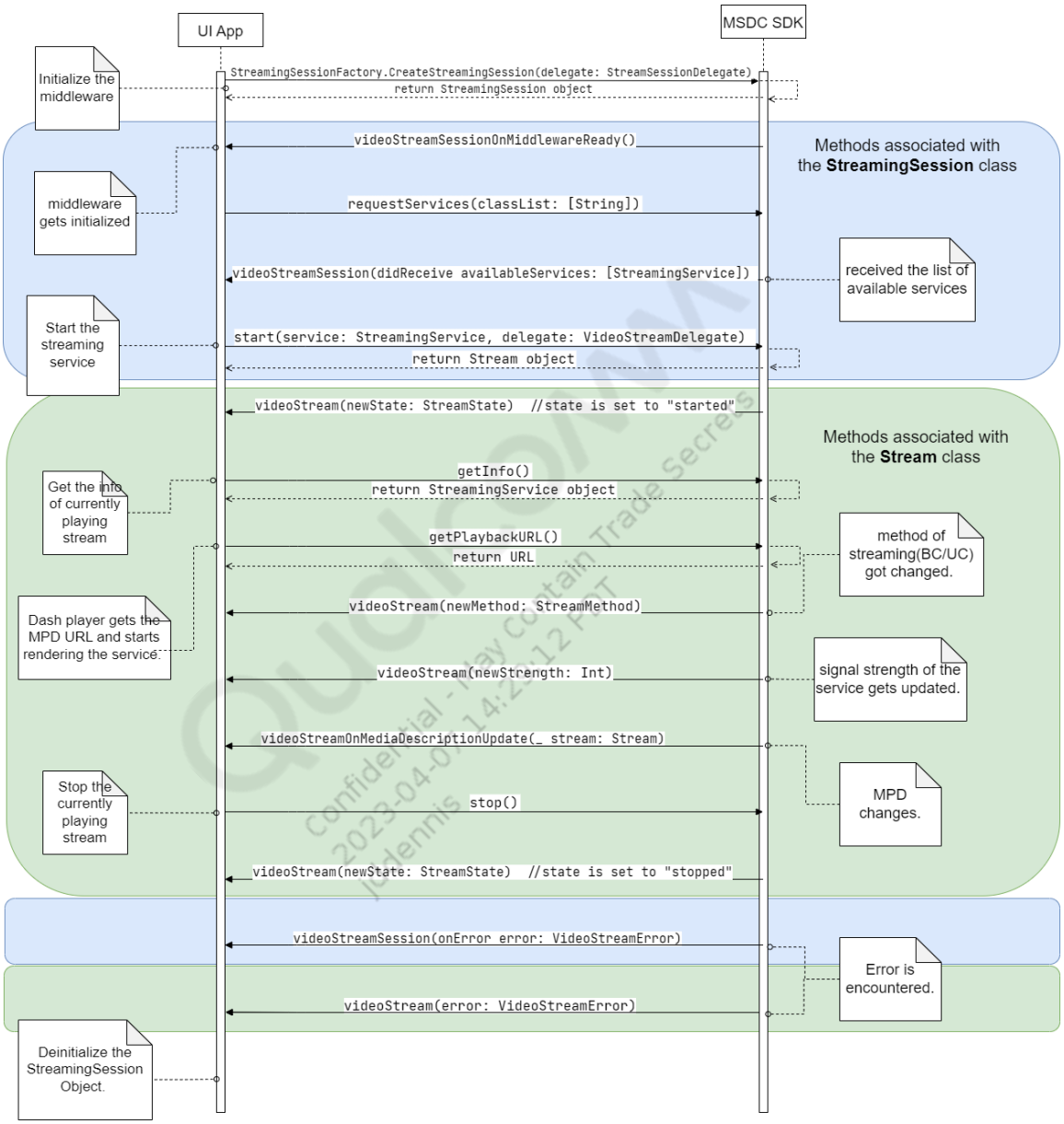
To support Streaming, the app must talk to the MSDC and DASH-enabled media player (see Figure 2-1). The app's communication with the MSDC is essentially a control path, while its communication with the media player is a data path.

3.1 Streaming service APIs

API	Description
`StreamingSessionFactory`	`CreateStreamingSession(delegate: StreamSessionDelegate):` This method initializes the streamingSession object and creates the object of the delegate class.
`StreamingSession`	The protocol that makes it possible for the application to stream media over MBMS, in accordance with the 3GPP specifications. ▪ `requestServices(classList: [String])` Obtains available services from the network. ▪ `start(service: StreamingService, delegate: VideoStreamDelegate)` → Stream The method takes the StreamingService object and the VideoStreamDelegate as parameters and plays the stream. It returns an object of type Stream. In case of error, it calls the delegate with the error message reason "Request Failed".
`StreamSessionDelegate`	The delegate protocol that is used to receive information from the middleware on MBMS streaming services. It can call the following methods: ▪ `videoStreamSession(onError error: VideoStreamError)` Called by the middleware when it has detected an error condition. Refer to VideoStreamError for more details. ▪ `videoStreamSessionOnMiddlewareReady()` Called by the middleware to indicate that the middleware has been initialized and is ready. ▪ `videoStreamSession(didReceive availableServices: [StreamingService])` Called by the middleware to indicate published Streaming Services have changed.

API	Description
`Stream`	The protocol that represents a specific Streaming service and contains the data and methods of the respective stream. It also reports feedback of the playing stream to the indicated callback(delegate). ▪ `stop()` Stops playing Streaming service. ▪ `getInfo()` Returns an object of the class StreamingService that contains data about the stream. ▪ `getPlaybackURL()` Returns the URL that can be used to play the corresponding Streaming service for which the method is called. The delegate (callback) for the Stream class through which the middleware will provide updates on the status of the stream.
`VideoStreamDelegate`	A delegate protocol to be used when the application is actively playing the Streaming service. The middleware updates the status of the stream via this callback. It calls the following methods: ▪ `videoStream(newMethod: StreamMethod)` Notifies the middleware if the broadcast/unicast method is being used currently. Refer the StreamMethod for more details. ▪ `videoStream(newState: StreamState)` Called to notify about the stream has changed state. Refer to the StreamMethod for more details. ▪ `videoStream(newStrength: Int)` Indicates that the broadcast signal strength is being updated. ▪ `videoStream(error: VideoStreamError)` Can be called by the middleware when it has detected an error condition in the current streaming service. Refer to VideoStreamError for more details. ▪ `videoStreamOnMediaDescriptionUpdate(_ stream: Stream)` Called to notify that the MPD of the currently playing stream has changed.
`StreamingService`	The struct that contains the data of a specific streaming service. It contains the following information: ▪ `serviceHandle`: Int → The number representing the service handle of the service. ▪ `sessionEndTime`: Double → The number of seconds from 00:00:00 UTC on 1 January 1970 till the time when the end time of the Stream is set. ▪ `sessionStartTime`: Double → The number of seconds since 00:00:00 UTC on 1 January 1970 till the time when the start time of the Stream is set. ▪ `serviceId`: String → The string representing the serviceID of the Stream. ▪ `serviceClass`: String → The service class associated with the Stream. ▪ `nameForLocale`: [String: String] → The languages in which the Stream is available. ▪ `locales`: [String] → Local code for the language.
`VideoStreamError`	Types of VideoStreamError: ▪ **Session**: Errors related to the session. ▪ **Service**: Errors related to the current service.
`StreamMethod`	The StreamMethod can be of the following types: ▪ **Unicast**: Files are transferred through HTTP. ▪ **Broadcast**: Content is transferred through MBMS to numerous devices at once.
`Streamstate`	The Streamstate can take the following values: ▪ **Stopped**: Represents the Stream in the end state. ▪ **Started**: Represents the Stream in running state. ▪ **Stalled**: Represents the Stream in stalled state.

3.2 Streaming service sequence



4 File Delivery service

This chapter describes the MSDC API descriptions and call flow sequences for an application that provides a File Delivery service to users.

The File Delivery service makes it possible to efficiently transfer files over the MBMS by utilizing the FLUTE protocol to transfer the files over multicast networks.

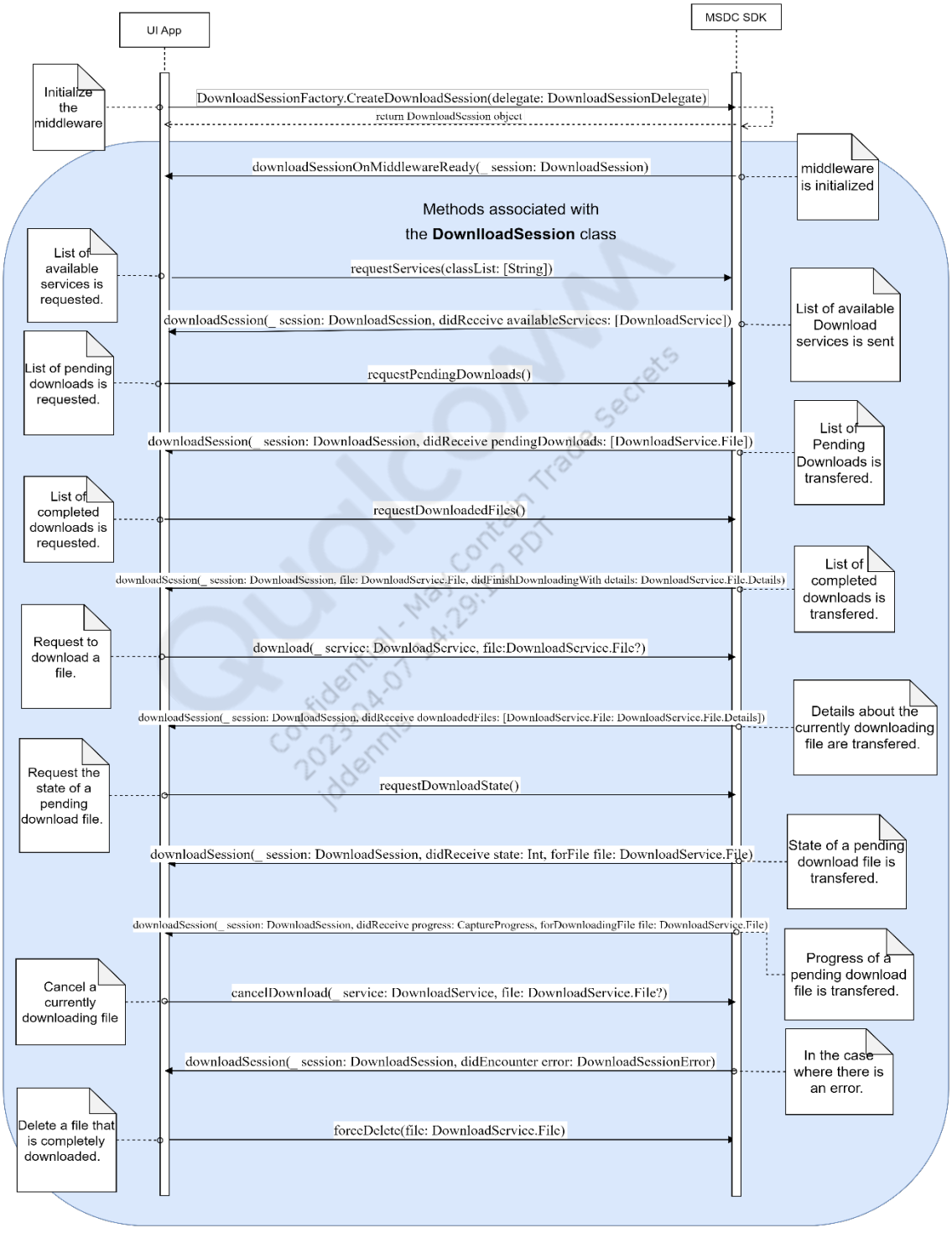
4.1 File Delivery service APIs

API	Description
`DownloadSessionFactory`	`CreateDownloadSession(delegate: DownloadSessionDelegate)` Method used to initialize the DownloadSession object and create the object of the delegate class.
`DownloadSession`	The protocol that controls the file download functionality over MBMS in accordance with the 3GPP specifications. 1. `requestServices(classList: [String])` Method that requests all available file download services. The available services are received through `availableServices()` called by the delegate. The classList is supposed to be provided as a parameter corresponding to which the services will be provided. 2. `requestPendingDownloads()` The pending files which are supposed to be downloaded are received by the delegate through `pendingDownloads()`. 3. `requestDownloadedFiles()` The requested files are downloaded (received) by the delegate through the downloadedFiles() method. 4. `download(_ service: DownloadService, file:DownloadService.File?)` The method that captures the entire file/service. Requests the MDM to begin the capture of a file or an entire service. If a single file is downloaded, it is specified as the second parameter, otherwise the entire service gets downloaded. The progress and updates 5. `cancelDownload(_ service: DownloadService, file: DownloadService.File?)` Cancels capture of a single file or complete service. Requests the MDM to stop the capturing of an individual file or a complete service. To stop capturing a single file, provide the file in the second argument. To stop capturing the complete service send null in the second argument. Make sure to call cancelDownload with the same arguments which were used when startDownload was called. 6. `requestDownloadState()` Requests information about the state of a file pending download. 7. `forceDelete(file: DownloadService.File)` Deletes the file forcefully.

API	Description
`DownloadSessionDelegate`	A callback class that apps should use to receive information on file downloads over MBMS. ▪ `downloadSession(_ session: DownloadSession, didReceive availableServices: [DownloadService])` Called when the list of available services gets updated. ▪ `downloadSession(_ session: DownloadSession, didReceive pendingDownloads: [DownloadService.File])` Called when the list of pending downloads is updated. ▪ `downloadSession(_ session: DownloadSession, file: DownloadService.File, didFinishDownloadingWith details: DownloadService.File.Details)` The parameters contain the downloaded file, the details of that specific file and the method is called when the file is finished downloading. ▪ `downloadSession(_ session: DownloadSession, didReceive downloadedFiles: [DownloadService.File: DownloadService.File.Details])` Called to transfer the information about the files/services fully downloaded. ▪ `downloadSession(_ session: DownloadSession, didReceive progress: CaptureProgress, forDownloadingFile file: DownloadService.File)` Called to transfer the information about the progress of the download of a specific file. Refer the CaptureProgress struct for more details. ▪ `downloadSession(_ session: DownloadSession, didReceive state: Int, forFile file: DownloadService.File)` Transfers the information about the state of a file pending download. ▪ `downloadSession(_ session: DownloadSession, didEncounter error: DownloadSessionError)` Indicates that the middleware has encountered an asynchronous error. Refer the dowloadSessionError struct fot more details. ▪ `downloadSessionOnMiddlewareReady(_ session: DownloadSession)` Called to indicate that the middleware has been initialized and is ready.
`DownloadService`	The struct that represents a specific download service and contains the relevant information about it. It contains the following data members: ▪ `serviceHandle`: Int →The number representing the service handle of the service. ▪ `sessionEndTime`: Double → The number of seconds from 00:00:00 UTC on 1 January 1970 till the time when the end time of the stream is set. ▪ `sessionStartTime`: Double → The number of seconds since 00:00:00 UTC on 1 January 1970 till the start time of the stream is set. ▪ `files`: [File] → The file that can be downloaded by the service. More details about the file struct can be found in the description of the **file** struct. ▪ `serviceId`: String → The string representing the serviceID of the stream. ▪ `serviceClass`: String → The service class associated with the stream. ▪ `nameForLocale`: [String: String] → The languages in which the stream is available. ▪ `locales`: [String] → Local code for the language.
`CaptureProgress`	The struct contains the following data members: ▪ `completedDownload`: int64 → Depicts progress of files that are completely downloaded. ▪ `totalDownload`: int64 → Total number of files to be downloaded. ▪ `completedDecode`: int64 → Represents decoding progress. ▪ `totalDecode`: int64 → Total number of files to be decoded.

API	Description
`dowloadSessionError`	The dowloadSessionError can assume the following values: ▪ `downloadFailure` ▪ `insufficientStorage` ▪ `fileDownloadProgressSuspended` ▪ `requestFailed`
`file`	The file struct contains the following data members: ▪ `uri`: String → URI of the file where it is stored. ▪ `httpUrl`: String → HTTP link of the location of the file. ▪ `md5`: String → Checksum string for verification. ▪ `fileSize`: int64 → Size of the file. ▪ `owner`: Owner → The owner to whom the file belongs.

4.2 File Delivery sequence



5 Group Call service

This chapter describes the MSDC API descriptions and call flow sequences for an application that provides a Group Call service to users.

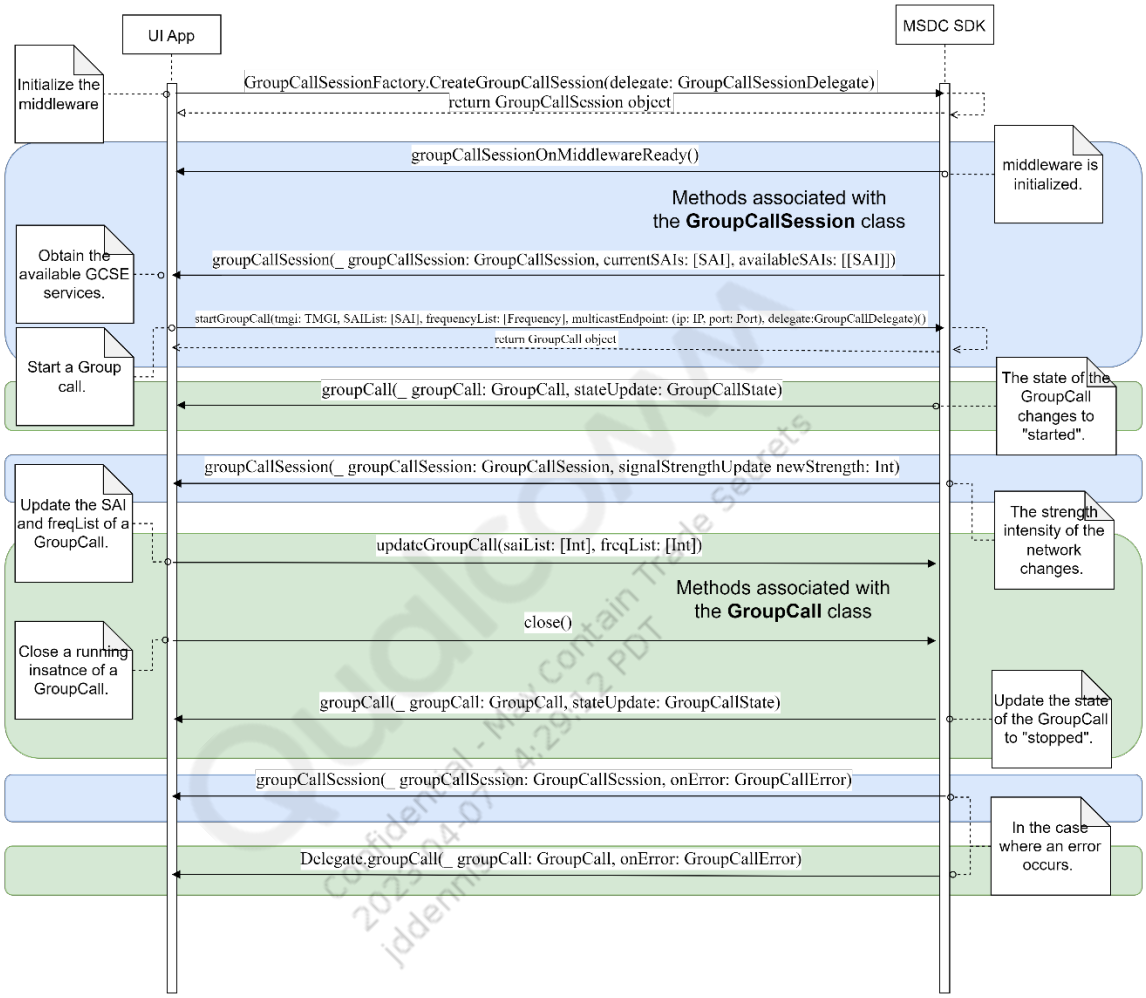
To support the Group Call service, the application must talk to the MSDC, Group Call Client, and Media Player. The app's communication with the MSDC is essentially a control path, while its communication with the Media Player is a data path. A group-call service requires the TMGI, the SAI list, and the frequency list to uniquely identify it among all the available Group Call services.

5.1 Group Call service APIs

API	Description
`GroupCallSessionFactory`	`CreateGroupCallSession(delegate: GroupCallSessionDelegate):` Method used to initialize the GroupCallSession object and create the object of the delegate class.
`GroupCallSession`	This class provides functionality for accessing group call functionality over MBMS in accordance with the 3GPP specifications. ▪ `startGroupCall(tmgi: TMGI, SAIList: [SAI], frequencyList: [Frequency], multicastEndpoint: (ip: IP, port: Port), delegate: GroupCallDelegate) -> GroupCall` Takes TMGI, SAIlist, Frequency list, IP port, and the groupcalldelegate as the parameters and creates the instance on a GroupCall object. This contains information about the group call service and can be used to control it. ▪ The delegate class for this method is used to receive the updates and new information from the middleware, such as if the SAIlist or frequency list gets updated, if an error has occurred, or if the middleware is ready and has been initialized correctly.
`GroupcallSessiondelegate`	A delegate protocol that is used to receive information from the middleware on MBMS group-call services. An instance of this object should be passed into GroupCallSession's startGroupCall() method. It can call the following methods: ▪ `groupCallSession(_ groupCallSession: GroupCallSession, currentSAIs: [SAI], availableSAIs: [[SAI]])` Indicates that the list of currently available SAIs has been updated. ▪ `groupCallSession(_ groupCallSession: GroupCallSession, onError: GroupCallError)` Called by the middleware when it has detected an error condition. Refer the GroupCallError class for more details. ▪ `groupCallSession(_ groupCallSession: GroupCallSession, signalStrengthUpdate newStrength: Int)` Indicates that the strength of the signal has been changed and the new strength of the network. ▪ `groupCallSessionOnMiddlewareReady(_ groupCallSession: GroupCallSession)` Called to notify that the middleware is initialized and running properly.

API	Description
`GroupCall`	▪ This protocol is used to represent a single eMBMS group call. After a call has been started with GroupCallSession's startGroupCall method, this class is used to hold information about the call and control it. ▪ It contains these data members: ▫ `Tmgi` Integer used as the unique identifier for a GroupCall. ▫ `State` The GroupCallState variable that stores the information about the state of the group call. Refer the GroupCallState for more details. ▪ The protocol contains these methods: ▫ `close()` Method that stops the currently playing group call. ▫ `updateGroupCall(saiList: [Int], freqList: [Int])` Sends an update to the middleware when the SAI (Service Area Identifier) list and frequency information of the group call has changed. It takes the SAIlist and the freqlist as the parameters. ▪ The delegate class (GroupCallDelegate) which can update the state of the current group call or notify if any error has occurred.
`GroupCallDelegate`	The callback class for use when the application is in a group call. The middleware will provide updates on the status of the call via this callback. ▪ `groupCall(_ groupCall: GroupCall, stateUpdate: GroupCallState)` Called when the state of the groupcall is changed. ▪ `groupCall(_ groupCall: GroupCall, onError: GroupCallError)` Called by the middleware when there is an error.
`GroupCallError`	The error GroupCallError can assume the following values: ▪ `Session`: Error related to the current session. ▪ `Service`: Error related to the service.
`GroupCallState`	The GroupCallState can assume the following values: ▪ `Started`: The service is in a running state. ▪ `Stopped`: The service has stopped. ▪ `Stalled`: The service has been paused. It can also contain a string where the reason for stalling can be mentioned.

5.2 Group Call service sequence



A References

A.1 Acronyms and terms

Acronym or term	Definition
MSDC	Multicast Service Device Client
SDK	Software Development Kit
FLUTE	File Delivery over Unidirectional Transport
DASH	Dynamic Adaptive Streaming over HTTP
eMBMS	Enhanced Multimedia Broadcast Mlticast Services
LTE	Long Term Evolution
MPD	Media Presentation Description
MVC	Model-View-Controller
SAI	Service Area ID
TMGI	Temporary Mobile Group ID
UE	User equipment
MDM	Mobile data modem