ETSI TS 126 347 V15.1.0 (2019-04)

LTE;
Multimedia Broadcast/Multicast Service (MBMS);
Application Programming Interface and URL
(3GPP TS 26.347 version 15.1.0 Release 15)

Reference
RTS/TSGS-0426347vf10

Keywords
LTE

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Technical Specification (TS) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities, UMTS identities or GSM identities. These should be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between GSM, UMTS, 3GPP and ETSI identities can be found under http://webapp.etsi.org/key/queryform.asp.

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

- Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

Introduction

The present document has been created as part of the MBMS Transport Protocol and API (TRAPI) work item and is based on the conclusions of TR 26.852 [6] in order to provide application-friendly methods and interfaces to access 3GPP MBMS User services. The present document is primarily targeted for developers of web and user applications and attempts to abstract complex MBMS procedures in simple methods and interfaces. MBMS Client vendors can implement this API and URL to simplify the integration of MBMS User Services.

1 Scope

The present document provides application methods and interfaces between an MBMS-aware application and the UE MBMS Client to access 3GPP MBMS User services. The purpose of the document is the definition of enablers in order to simplify the usage of MBMS in web-centric as well as app-based service environments.

The present document defines several APIs to access MBMS User Services and a URL to access resources available as part of an MBMS User Service. The MBMS User Services are defined in TS 26.346 [5] and are not part of the present document.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same* Release as the present document.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 22.146: "Multimedia Broadcast/Multicast Service; Stage 1".

[3] 3GPP TS 22.246: "Multimedia Broadcast/Multicast Service (MBMS) user services; Stage 1".

[4] 3GPP TS 23.246: "Multimedia Broadcast/Multicast Service (MBMS); Architecture and functional description".

[5] 3GPP TS 26.346: "Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs".

[6] 3GPP TR 26.852: "Multimedia Broadcast/Multicast Service (MBMS); Extensions and profiling".

[7] 3GPP TS 26.247: "Transparent end-to-end Packet-switched Streaming Service (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH)".

[8] IETF RFC 2616: "Hypertext Transfer Protocol -- HTTP/1.1".

[9] Object Management Group: "Interface Definition Language™ (IDL™) 4.0".

[10] IETF RFC 3066: "Tags for the Identification of Languages".

[11] IETF RFC 3986: "Uniform Resource Identifier (URI): Generic Syntax".

[12] 3GPP TS 29.116: "Representational state transfer over xMB reference point between content provider and BM-SC".

[13] IETF RFC 7595: "Guidelines and Registration Procedures for URI Schemes".

[14] IETF RFC 7230: " Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing".

[15] IETF RFC 7553, "The Uniform Resource Identifier (URI) DNS Resource Record"

[16] IETF RFC 6335, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry"

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in TR 21.905 [1].

Application User Service: A service for which all associated resources are delivered through an MBMS User Service including broadcast and unicast and which is accessible through the MBMS-API.

MBMS-API: An Application Programming Interface between the MBMS Client and the MBMS-Aware application for a specific Application User Service.

MBMS-aware Application: An application in the user space that communicates with the MBMS client through APIs as defined in the present document.

MBMS Client: A function that implements functionalities defined in TS 26.346 and provides APIs and protocol-related methods to expose relevant functionalities to an MBMS-aware application.

MBMS-URL: A Universal Resource Locator that enables a general application to access resources delivered through an MBMS User Service using the MBMS URL handler.

MBMS-URL Handler: A logical function that translates the MBMS-URL to a sequence of MBMS-API calls in order to provide resources delivered by MBMS User Services to a general application.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in TR 21.905 [1].

API	Application Programming Interface
DANE	DASH-Aware Network Element
DASH	Dynamic Adaptive Streaming over HTTP
IDL	Interface Definition Language
JSON	JavaScript Object Notation
MAA	MBMS-Aware Application
MPD	Media Presentation Description
SAND	Server and Network Assisted DASH
SDP	Session Description Protocol
USD	User Service Description
URL	Universal Resource Locator

4 Overview

4.1 Introduction

The present document addresses a specific interface for an end-to-end application service, namely the interface between the MBMS client and the MBMS-Aware Application (MAA) as shown in Figure 4.1-1. An application service provider may provide content through xMB (see TS 26.346 [5] and TS 29.116 [12]) to the BMSC, but may also provide information directly to an MAA. The BMSC uses MBMS User Services as well as MBMS bearer services and unicast bearers to communicate with the MBMS client. The present document deals with the interface between the MBMS client and the MAA, referred to as MBMS Application Programming Interfaces (MBMS-APIs). The APIs may be used directly by the MAA, or the MBMS URL Handler may use the MBMS-APIs after receiving an MBMS URL from a generic application.

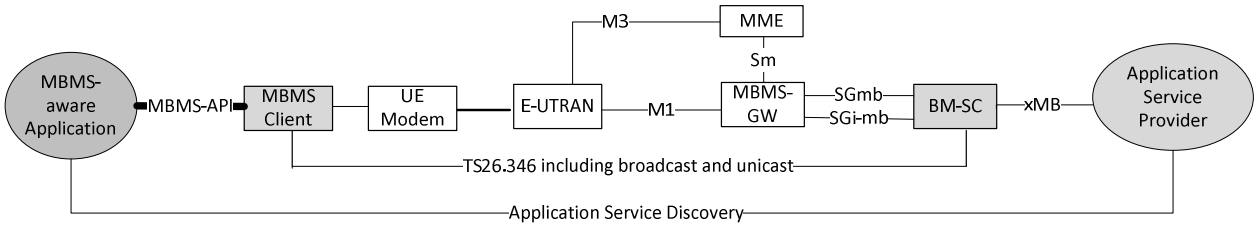

Figure 4.1-1: End-to-end Architecture for Application Service Providers using eMBMS for Delivery

4.2 Network Architecture and MBMS User Services (Informative)

According to TS 26.346 [5], three distinct functional layers are defined for the delivery of an MBMS-based service:

1) **Bearers**: Bearers provide the mechanism by which IP data is transported. MBMS bearers as defined in 3GPP TS 23.246 [4] and 3GPP TS 22.146 [2] are used to transport multicast and broadcast traffic in an efficient one-to-many manner and are the foundation of MBMS-based services. MBMS bearers may be used jointly with unicast PDP contexts in offering complete service capabilities.

2) **Delivery Method**: When delivering MBMS content to a receiving application one or more delivery methods are used. The delivery layer provides functionality such as security and key distribution, reliability control by means of forward-error-correction techniques and associated delivery procedures such as file-repair, delivery verification. Three delivery methods are defined, namely download, streaming, transparent and group communication. The present document does not address group communication.

3) **User service**: The MBMS User service enables applications. Different applications impose different requirements when delivering content to MBMS subscribers and may use different MBMS delivery methods.

MBMS User Service architecture is based on an MBMS client on the UE side and a BM-SC on the network side. Details about the BM-SC functional entities are given in figure 4 of TS 26.346 [5].

The BM-SC and UE may exchange service and content related information either over point-to-point bearers or MBMS bearers whichever is suitable. Among others, the following MBMS procedures are defined in TS 26.346 [5]:

- User Service Discovery / Announcement providing service description material to be presented to the end-user as well as application parameters used in providing service content to the end-user.
- MBMS-based delivery of data/content from the BM-SC to the UE over IP multicast or over IP unicast.
- Associated Delivery functions are invoked by the UE in relation to the MBMS data transmission. The following associated delivery functions are available:
  - File repair for download delivery method used to complement missing data.

4.3 MBMS Application User Services

4.3.1 Introduction

The MBMS system may provide services to an MAA for which all associated resources are delivered through an MBMS User Service including broadcast and unicast. The services may be made accessible through an MBMS-API defined in the present document. The present document defines an initial set of Application User Services and corresponding MBMS-APIs. For each Application User Services, one MBMS-API is defined.

MBMS Application User Services define how an MAA gets access to the content delivered via MBMS user services.

The specification may be extended to add additional Application User Services and MBMS-APIs.

The MBMS Application User Services that are covered by the present document are defined in this clause.

4.3.2 File Delivery Application User Service

The File Delivery Application User Service provides MAAs with methods to manage the reception of files delivered over MBMS Download Delivery services as defined in TS 26.346 [5]. Some of the defined interfaces allow an MBMS-Aware Application to get information on the available MBMS File Delivery Application Services and possibly on the files scheduled to be carried on these services; to start and stop the capture of files on these services; and to allow the MBMS Client to provide notifications associated with the reception of files. Clause 6.2 provides a complete description and the associated uses for the interfaces in the File Delivery Application Service API and includes an abstract IDL definition for these interfaces.

4.3.3 DASH Application User Service

The DASH Streaming Service API defined in clause 6.3 provides MAA with interfaces to manage the reception of DASH streaming content (as defined in TS 26.247 [7]) delivered over DASH-over-MBMS Streaming Services (as defined in TS 26.346 [5], clause 5.6). Some of the interfaces defined allow an MAA to get information on the available DASH Streaming Services; to start and stop the reception of DASH streaming content on these services; and to allow the MBMS Client to provide notifications associated with the receptions of DASH streaming content. Clause 6.3 provides a complete description and the associated uses for the interfaces in the DASH Streaming Service API and also includes an abstract IDL definition for these interfaces.

4.3.4 MBMS RTP Streaming User Service

MBMS RTP Streaming User Service provides the MAA with interfaces to access MBMS Streaming Delivery Services as defined in TS 26.346 [5]. The MAA may request start or stop any available RTP streaming service. The MAA will receive information about the RTP data. The API for the MBMS RTP Streaming User Services is defined in clause 6.4 using the the `serviceType` set to `RTP` for the service request.

The SDP provided in the `sdpURI` should be used together the RTP interface as documented in clause 7.5.

4.3.5 MBMS Transparent User Service

MBMS Transparent User Service provides the MAA with interfaces to access MBMS transparent delivery services as defined in TS 26.346 [5]. The MAA may request start or stop any available transparent service.

The Service API is defined in clause 6.4 using the the `serviceType` set to `TRANSPARENT` or `TRANSPARENT-ROM` for the service request.

`TRANSPARENT` refers to any transport service that is declared as a transparent service by the BMSC, if the Service Announcement includes a required capability `'24'`, i.e. the signal for the “MBMS User Service Discovery / Announcement Profile for Transparent Delivery Services” as documented in Annex L.5 of TS26.346 [5].

`TRANSPARENT-ROM` refers to any transparent service that is also a Receive-Only Mode (ROM) Service, if the ROM service is signalled in the User Service Description.

The SDP provided in the `sdpURI` should be used together the Packet Data interface as documented in clause 7.6.

4.4 Specification Outline

The following aspects are defined in the present document:

- The definition of different interfaces as part of client reference architecture in clause 5.
- A set of service APIs (MBMS-APIs) for each application user service as defined in clause 4.3. The definition provide the ability to independently develop MBMS-aware applications and MBMS clients, even for different operating systems and execution environments, but rely on the service APIs to communicate with the MBMS client and to make use of the MBMS functionalities. This is defined in clause 6.
- A set of interface options between the MBMS client and the application to support the transfer of user data. Primary focus is on the communication through HTTP interfaces, as for example DASH or other application services. This is defined in clause 7.
- The mapping of the functionality to a URL handling and abstraction of the services in such environments. This is defined in clause 8.

5 Reference Client Architecture

Figure 5.1 shows a general service architecture including a reference client. On the network side, an MAA and content provider provides media formats to a BM-SC, typically through the xMB interface and initiates services and sessions through the xMB interface. The BMSC establishes MBMS User Services and the lower layers support the delivery of the data through regular 3GPP unicast as well as MBMS broadcast bearers.

The MAA initiates the communication with the MBMS client using the MBMS-API. The MBMS client identifies the relevant services and provides the delivered user data to MAA. Typically, the media formats are provided through interfaces and to functions that conform to well-defined media delivery formats. The MAA controls the media client.

In TS 26.346 [5] the interface between the BMSC and the MBMS client for both unicast and broadcast related services and functions are defined. The interface between the MBMS client and the MAA is defined in the present document..

The focus of the present document is to define app-developer and web-friendly interfaces and programming structures to enable such an MAA to interact with the MBMS client.

An MAA that communicates with the MBMS client through APIs and possibly URL handlers as defined in the present document is referred to as MAA**.** The MBMS Client is a function that implements functionalities defined in TS 26.346 and provides APIs interfaces to expose relevant functionalities to an MAA.

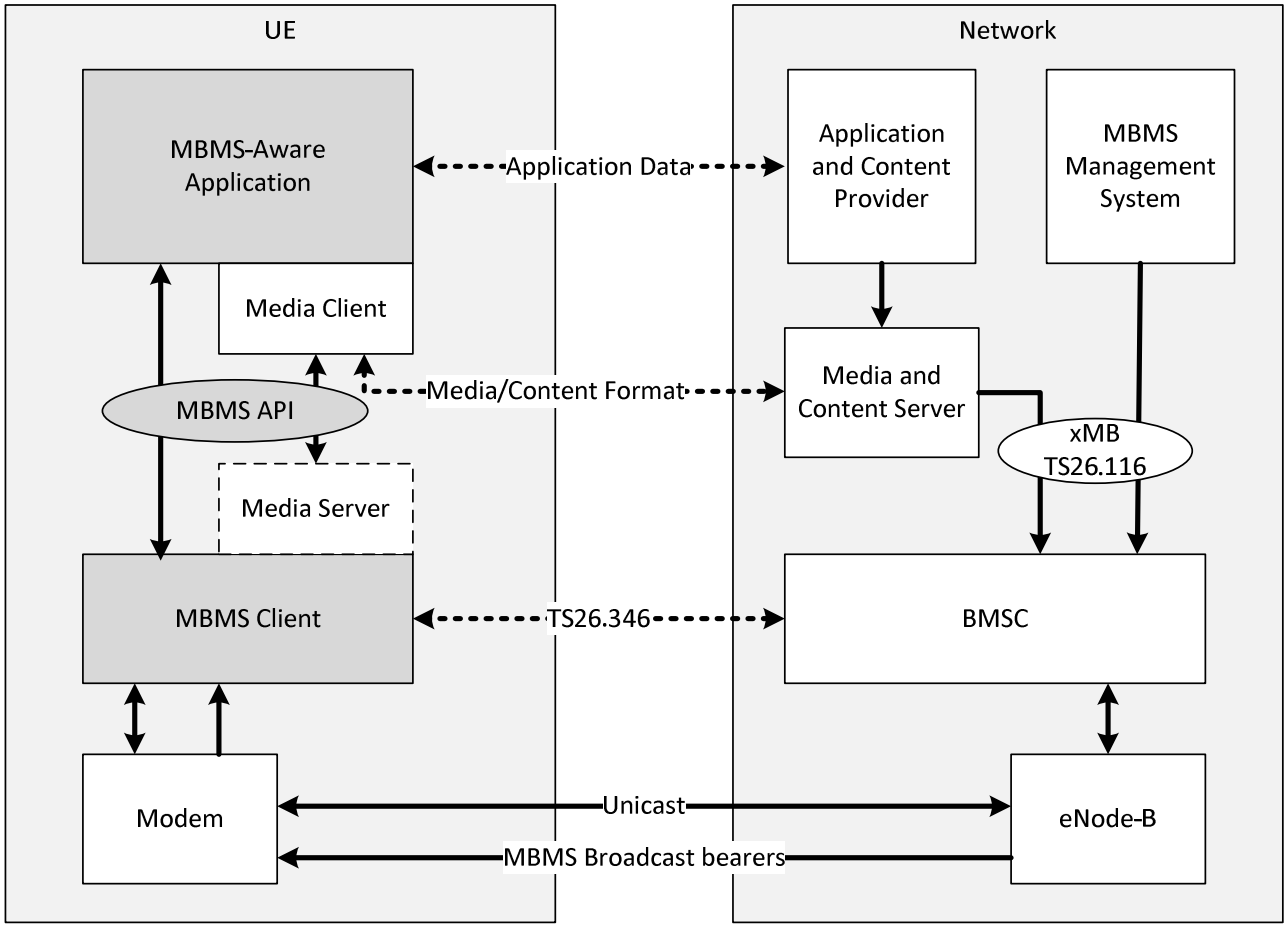

Figure 5.1: General Reference Architecture for Client

Figure 5.2 shows a specific instantiation for the DASH Streaming Application User Service using the DASH Streaming MBMS-API. In this case the content formats conform to TS 26.247. The DASH client may be viewed as part of the MAA, and as 3GPP defines interfaces into a DASH client in TS 26.247 [7] also interfaces to the DASH client function are in focus of the present document.

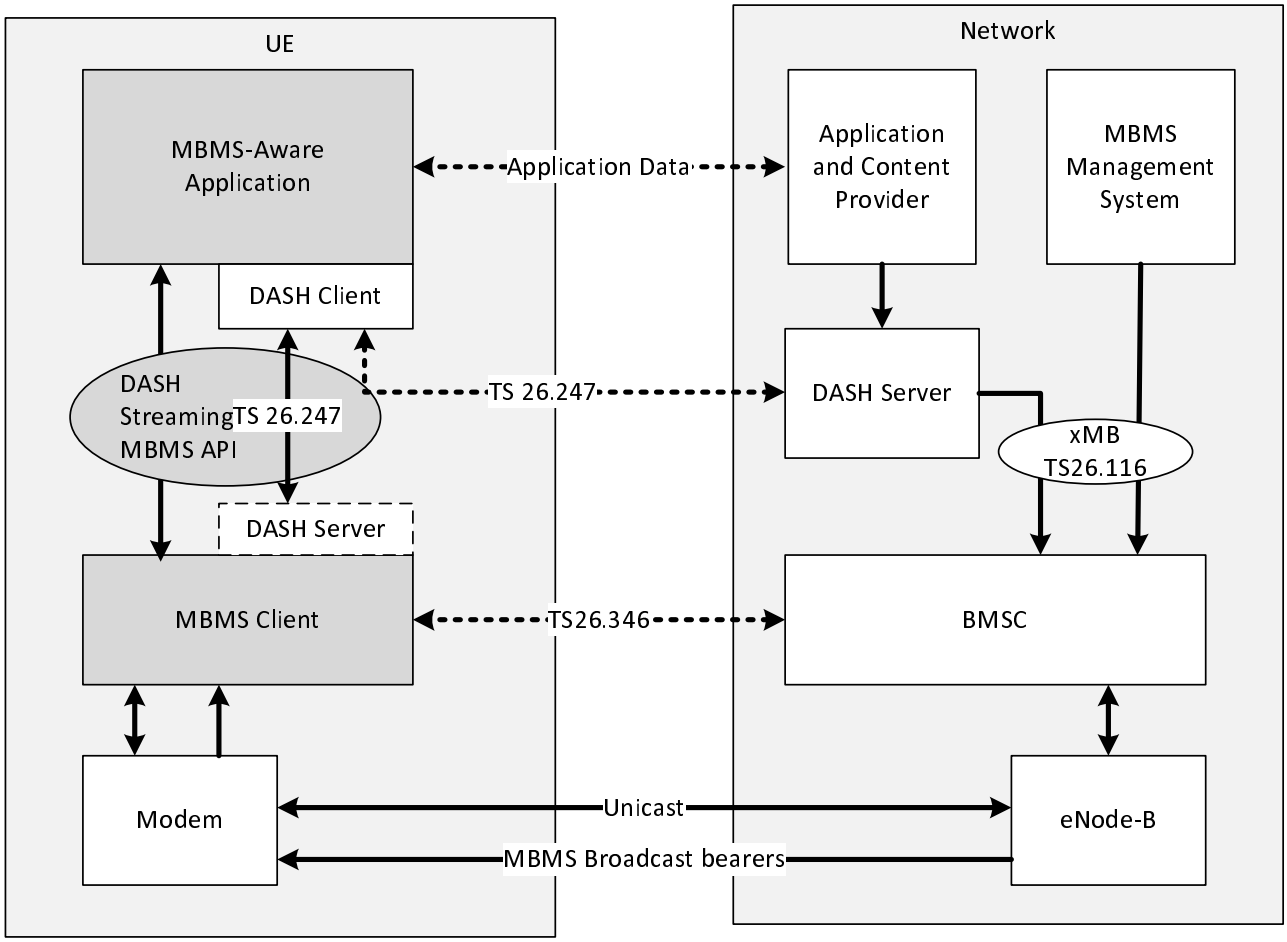

Figure 5.2: Client Reference Architecture for DASH-over-MBMS Streaming

The control part of the MBMS-API are used for service discovery, registration, notifications, state changes and other control messages between the MAA and the MBMS client and for other control messages. The control part of the MBMS-APIs are defined in clause 6.

The data part interfaces are used to provide content delivered through MBMS User services to the MAA. However, the data is using formats and interfaces that are primarily MBMS independent such that for the MAA the delivery over MBMS is obscured. MBMS Client to application interfaces for data is primarily introduced in clause 7.

6 MBMS Application Programming Interfaces

6.1 Overview

6.1.1 Background

Figure 6.1.1-1 provides a graphical overview of how the MBMS Control Application Programming Interface (API) fits into the UE architecture of delivering MBMS content to MAAs.

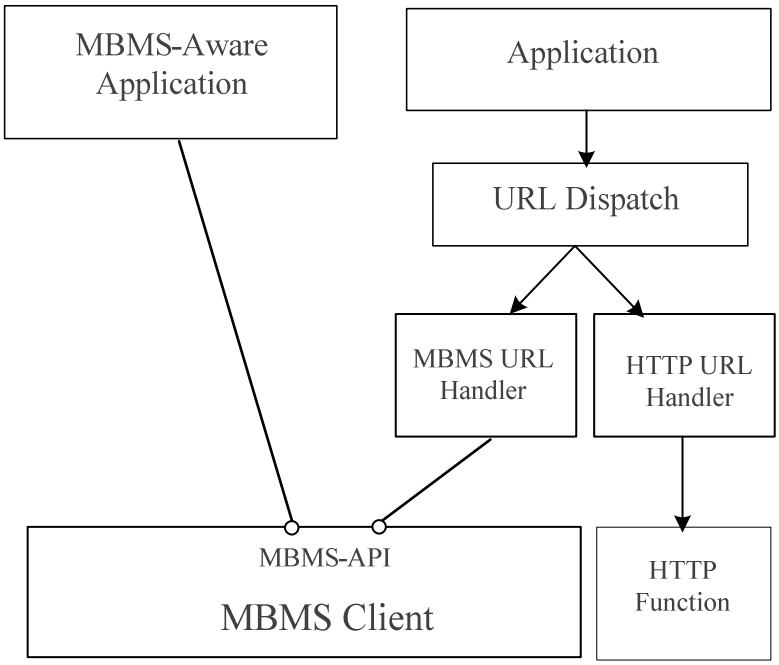

Figure 6.1.1-1: MAA to MBMS function API

The MBMS API implementation on High-Level Operating Systems and application development frameworks is typically realized as a programmatic library that is linked to the application code and that runs in the application context. That library implementation communicates with a particular MBMS Client implementation and abstracts the implementation-specific interactions with the MBMS Client from the MAA. The MBMS-API exposes to the MAA a set of simple interfaces described in the IDL definitions in clauses of the present document; in particular, the IDL makes use of callback functions as the means for the MBMS Client to notify MAAs of events relevant to the reception of content delivered over MBMS user services. The programmatic library communication with the MBMS Client is implementation-specific, it is not in the scope of the present document, and it can be implemented using different solution approaches (e.g., smartphone High-Level Operating System services, WebSockets, etc.).

It is understood that in some application development frameworks (e.g., HTML/Web Applications), linking a programmatic library to the MAA is not possible. In these cases, the callback functions may not be realized programmatically as function calls. In particular, the MAA may need to implement the necessary approach available on these frameworks (or the selected solution approach) to receive event notifications from the MBMS client in place of callback functions. For such frameworks, the implementation of callback functions described in the IDL of the present document is not required. However, the information structures defined on the IDL callback functions are to be communicated to the MAA when the MBMS Client generates the corresponding event notification to the MAA using the available (or selected) notification mechanism.

Figure 6.1.1-2 provides an overview of the graphical representation of multiple MAAs connecting to the MBMS client. Each MAA uses its own instantiation of the MBMS-API.

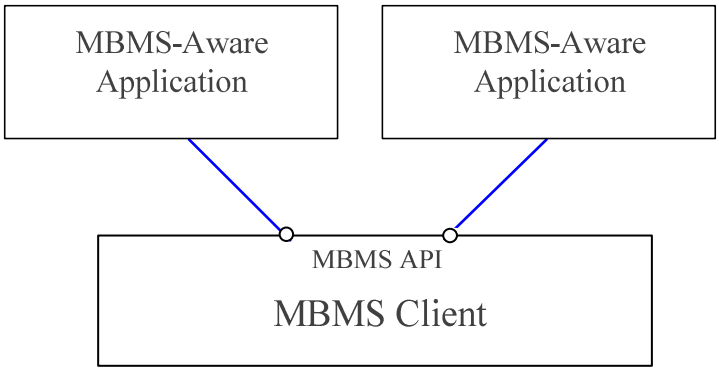

Figure 6.1.1-2: Multiple MAAs connecting to MBMS Client

Guidelines on API documentation are provided in Annex A.

It is also recommended to separate the description for MBMS client functions and the description of the MAA functions as it is expected that they are developed by independent parties.

6.1.2 Parameter description notation

In each Application Service API described in this clause, the parameters in the API interfaces are described in the following format:

- **`dataType`** `parameterName` – description of the parameter.

Each interface parameter is defined via a separate item in the list of parameters. The **`dataType`** on the item defines the programmatic data type for the parameter named `parameterName`, as described on the IDL for the API; the **`dataType`** may be a complex structure on the IDL. The `parameterName` provides a name for a parameter on the API interface. A description follows to provide information on the meaning and use for the parameter.

6.1.3 MBMS Client State Model

For each Application Service, the MBMS client maintains certain states for the application and a set of internal parameters. The internal parameters may be updated based on information received from the MBMS system, by time-outs, or by communication with the application. It is recommended that an MBMS client state model for each defined MBMS-API is defined.

6.2 File Delivery Application Service API

6.2.1 Introduction

The File Delivery Application Service API provides MBMS Aware applications with interfaces to manage the reception of files delivered over File Delivery Application User Services. This API is intended to support applications that are running while files are being delivered through MBMS user services as well as applications that are not running to receive information on files received through an MBMS User service, for example as the user may have quit/exited the application.

In order to support applications that may not be currently running while files are being received, the MBMS client may keep received files for a period of time configured by the application, which includes means to collect received files even if the user does not actively interact with the application to consume the received files.

When the application is currently running and can collect the files received over MBMS User services, the MBMS client moves the files to the application space. It is ultimately the application's responsibility to manage the storage of requested files, especially the amount of storage to be used.

Any persistent storage of received files by the MBMS client is only intended to ensure that the received files are made available to the respective requesting application. Once files are moved/copied to the application space, the application is responsible for managing those files.

The IDL for the File Delivery Application Service API is defined in clause B.2.

6.2.2 MBMS Client State Model

6.2.2.1 Overview

Figure 6.2.2.1-1 provides an informative MBMS client state model in order to appropriately describe the messages on the service API. Five different states are defined as listed in Table 6.2.2.1-1.

Note that the state model is defined on the granularity per service and each MAA. This means that the MBMS client may at least conceptually have to maintain multiple state machines, namely one for each MAA/service combination.

The state model does not imply any implementation requirements for an MBMS client, but is used as a model to support the description of the APIs.

State changes may be the result of:

- Requests from the MAA
- Timer expiration in the MBMS client

- Information provided by the MBMS User Service (USD, schedule, FDT, file complete)

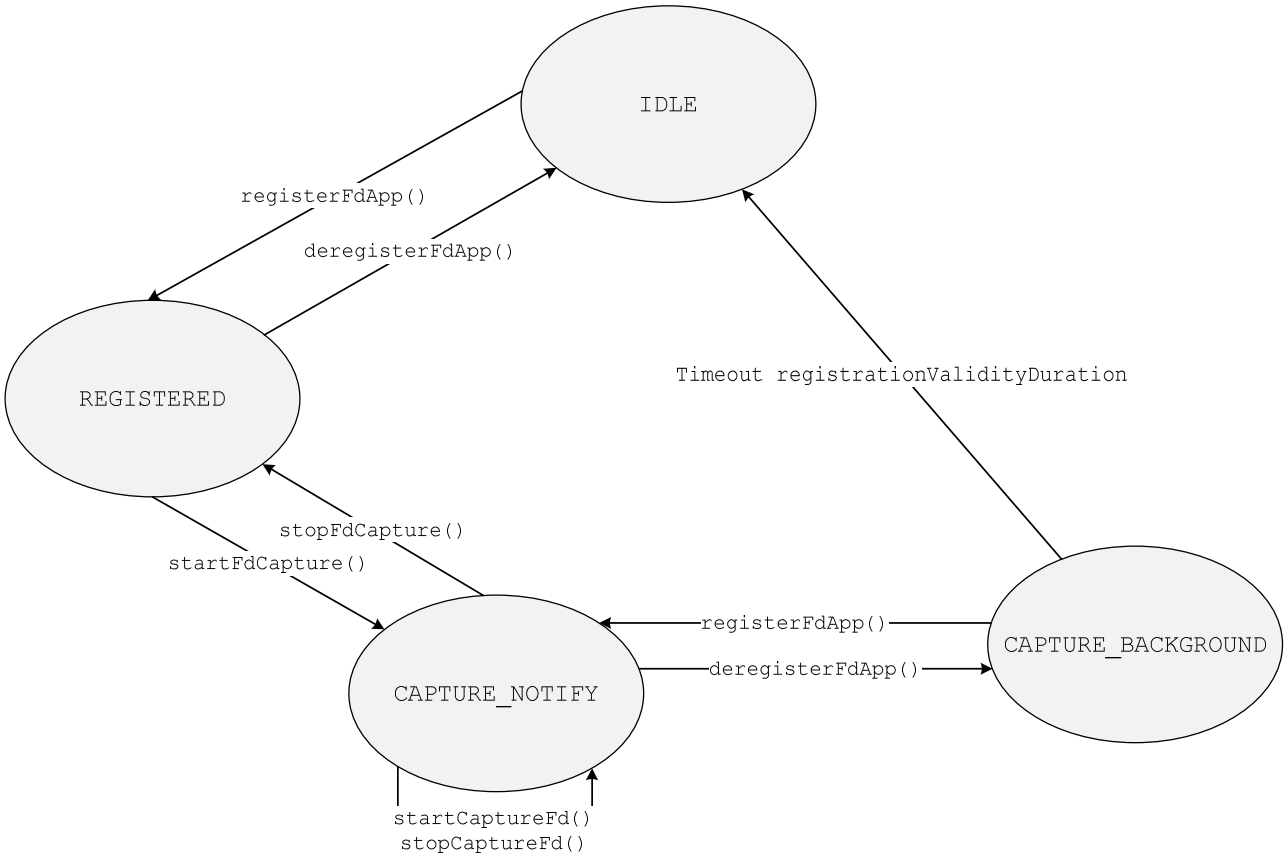

Figure 6.2.2.1-1: State Diagram

Table 6.2.2.1-1 describes the states for the MBMS client. Detailed descriptions are provided in the following subclauses.

Table 6.2.2.1-1: States and Parameters of MBMS Client

States and Parameters	Definition
IDLE	In this state the MAA is not registered with the MBMS client and it may not keep the service parameters up-to-date. For more details see clause 6.2.2.3.
NON_AVAILABLE	In this state the MBMS client is not available and an MAA cannot register with the MBMS client. The MBMS client may not be aware of the state, it is only a state perceived by the MAA.
REGISTERED	In this state the MBMS client has registered the MAA, it may keep the service definition up to date, but it is not providing file capture services to the MAA(s). For more details see clause 6.2.2.4.
CAPTURE_NOTIFY	In this state the MBMS client provides file capture for one specific service to a registered MAA and notifies the MAA on any received file. For more details see clause 6.2.2.5.
CAPTURE_BACKGROUND	In this state the MBMS client provides file capture for one specific service to an MAA without notifying the MAA on any received file for some agreed time. For more details see clause 6.2.2.6.

6.2.2.2 MBMS Client Internal parameters

The MBMS client maintains internal parameters as defined Table 6.2.2.2-1. Note that the parameters are conceptual and internal and only serve for the purpose to describe message generation on the API calls.

Table 6.2.2.2-1: Parameters of MBMS Client

States and Parameters	Definition
_Preconfigurations	
_maxRegistrationValidityDuration	MBMS client parameter that provides the maximum time after deregistration that a client still captures files on behalf of an MAA.
_defaultAvailabilityDeadline	the time a file that is kept in the MBMS client owned storage location.
_app[]	The MBMS client maintains a parameter set per registered app
_appId	A unique ID provided by the MAA and assigned to the app.
_serviceClass[]	A list of service classes identifying the services the MAA has access to.
_registrationValidityDuration	A period of time following the MAA de-registration over which the MBMS client continues to capture files for the MAA.
_locationPath	The storage location where the MAA wants the MBMS client to store collected files.
_backgroundCaptureTimer	Timer on how long files are captured in background.
_service[]	The MBMS client maintains a parameter list per service. In this context the list is assigned also to one app, but an implementation may share the internal parameter list assigned to a service across multiple apps.
_serviceID	The service ID for a File Delivery Application service over which the MBMS client collects files for the MAA.
_serviceClass	The service class associated with the File Delivery Application service assigned the Service ID..
_serviceLanguage	The language of the service
_serviceName[] _name _lang	The service name, possibly expressed in different languages.
_serviceBroadcastAvailability	The service broadcast availability for the client.
_fileCaptureRequest[] - _fileURI - _disableFileCopy - _captureOnce	A sequence of requested file captures from the MAA with some parameters. The _fileURI identifies the file(s) as they will also be used on the FLUTE FDTs for the File Delivery Application Service Allowed values for the _fileURI include: - The empty string signals that the MAA is interested in receiving all new files and updates to previously received files. - A BaseURL, i.e., a complete path for subdirectory (a prefix) identifying a group of files under that directory. - An absoluteURL, i.e., a complete URL that identifies a single file resource. _disableFileCopy – when set to true, this signals that the MAA does not want the MBMS client to make the file available on the application space. _captureOnce – when set to true, signals that a file matching a requested via the _fileURI (i.e. a file matching a BaseURL in fileURI, or any file if the fileURI is empty) is to be captured only once.

States and Parameters	Definition
_fileURIStatus[] _URI _contentType _deliveryState _md5 _deliverySchedule[] _start _stop _appState _notified _fileLocation	A sequence of _fileURIStatus[] that states the status of files the MAA may be interested or the service provides, possibly using scheduling information. The internal parameters store relevant information on the file such as content_type or MD5, if the information is available yet. The delivery state can be: - FD_RECEIVED - the file is received by the MBMS client and no more updates are expected according to the file schedule. - FD_SCHEDULED - scheduled, if a file schedule is delivered and a file is scheduled, but not yet received. - FD_IN_PROGRESS – the reception of the file is in progress The delivery schedule records the current upcoming delivery schedules. The application state may be transitioned to express that - the file was moved to the application, - failed if the move failed or - the location path points to an MBMS client controlled space. A _notified flag is provided to indicate if the MAA has been notified on the reception of the file.
_sessionSchedule[] _start _stop	Documents the session schedule for this session. Only _sessionSchedule records should be included for which the value of the _stop time is in the future.

6.2.2.3 MBMS Client Operation in IDLE state

In the IDLE state, the MBMS client may listen to the User Service Bundle Description and may collect information. However, no binding with the MAA is in place.

When the MBMS client received the registerFdApp() call as defined in subclause 6.2.3.2, then:

1) The MBMS client checks the input parameters for consistency and sets the internal variables

   a) If appId is an empty string then the MBMS client throws a MISSING_PARA

   METER result code in the registerFdResponse() as defined in subclause 6.2.3.3 and aborts these steps. If not, the MBMS client sets the internal variable _appId to the value of the parameter.

   b) The MBMS client adds each entry in the serviceClassList parameter to its _serviceClass[] record. Note that the serviceClassList parameter may contain an empty service class entry. If an empty service class is provided, the MBMS client considers the MAA to be registered with a service class that is also empty and only allows the MAA to have access to File Delivery Application Services that are not associated with a serviceClass (i.e., the USD for these services do not have a serviceClass defined).

   c) On receiving a registerFdApp() following a deregisterFdApp(), the MBMS client updates the serviceClassList to its _serviceClass[] record in the same way described for the setFdServiceClassFilter() method.

   d) If locationPath is not defined, the MBMS client provides means such that the MAA has access to the files the MBMS Client received on behalf of the MAA. If the locationPath is defined, the internal variable _locationPath is set to the value of the parameter.

   e) If registrationValidityDuration is not defined, the value of the internal parameter _registrationValidityDuration is set to 0 (zero) . If the registrationValidityDuration is defined, the internal variable _ registrationValidityDuration is set to this parameter, or to _maxRegistrationValidityDuration if the value of registrationValidityDuration is larger than _maxRegistrationValidityDuration.

f) If `callBack` is defined, the MBMS client uses the interfaces in the callback parameter of the `registerFdApp()` interface to send notifications of event occurrences to the MAA.

2) generates a response `registerFdResponse()` as defined in subclause 6.2.3.3 and changes to `REGISTERED` state:

a) If the MBMS client functions cannot be activated for any reason, especially if the File Delivery Application Service API did not find an MBMS client available on the UE on which the MAA is running, the `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE` registration response code is sent. The MBMS client may provide a `message` and may set `acceptedFdRegistrationValidityDuration` to any arbitrary value.

b) If the MAA did not provide a mandatory parameter, the MBMS client functions cannot be activated and the `MISSING_PARAMETER` registration response code is sent. The MBMS client may provide a `message` and may set `acceptedFdRegistrationValidityDuration` to any arbitrary value.

c) If the MBMS client functions can be activated, then

i the `RegResponseCode` is set to `REGISTER_SUCCESS` registration response code,

ii a `message` may be generated,

iii the MBMS client sets the value of `acceptedFdRegistrationValidityDuration` to the `_registrationValidityDuration`.

The MBMS client sends the response with the above parameters

3) If the MBMS client functions can be activated and the response is sent with a `REGISTER_SUCCESS`, then MBMS client is in `REGISTERED` state and uses the `REGISTERED` parameters to provide the list of matching file delivery services using the information in the User Service Description (USD). If the response is sent with a `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE`, then the MBMS client is in `NOT_AVAILABLE` state. If the response is sent with a `MISSING_PARAMETER`, then MBMS client is in `IDLE` state.

If the MBMS client receives the `getVersion()` API call, it shall return version `"1.0"`.

6.2.2.4 MBMS Client Operation in REGISTERED state

For each registered MAA and the assigned parameters according to Table 6.2.2.2-1, the MBMS client uses the information in the User Service Description as well as its internal state information for the MAA in `_app[]` in the service class list `_serviceClass[]` to collect and keep up-to-date all internal information for the services of interest for the app, i.e. those that are member of any service class for which the MAA has interest.

For each MBMS user service for which the USD as defined in TS 26.346 [5] is available in the MBMS client for the service classes registered by the MAA in `_serviceClass[]`, one service record in the internal parameter `_service[]` is defined in the MBMS client and continuously updated whenever an updated USD is available:

- For each **`userServiceDescription.name`** element, a (`name`, `lang`) pair is generated and added to the `_serviceName[]` list with `_name` set to the value of the USD element, and if present, the `_lang` set to the value of the associated `@lang` attribute. If no `@lang` attribute is present, the `_lang` parameter is set to an empty string.

- If the attribute **`userServiceDescription`**`@serviceClass` is present, the value of this attribute is assigned to `_serviceClass`. If not present, the `_serviceClass` is set to an empty string.

- The value of the attribute **`userServiceDescription`**`@serviceId` is assigned to `_serviceId`.

- If the attribute **`userServiceDescription`**`@serviceLanguage` is present, the value of this attribute is assigned to `_serviceLanguage`. If not present, the `_serviceLanguage` is set to an empty string.

- The `_serviceBroadcastAvailability` is continuously updated set it to `BROADCAST_AVAILABLE`, if broadcast is available (if the UE is in broadcast coverage of the service), if not, it is set to `BROADCAST_UNAVAILABLE` (if the UE is NOT in broadcast coverage of the service).

- If the **userServiceDescription.schedule** element is present then the MBMS client uses the information in the schedule description fragment to generate the internal _fileURI[] and _sessionSchedule[] list and keep up to date as a result of USD updates. The MBMS client should only include _fileUri[] list if there is a current or a future scheduled transmission of that file. The MBMS client shall only include _sessionSchedule[] records if the _stop value is in the future. The _deliveryState is set appropriately, e.g. to FD_SCHEDULED if the file is scheduled and FD_IN_PROGRESS, if the FDT is available, but the file reception is not yet completed.

If updates are provided and added to the _service[] parameter, the MBMS client should send a fdServiceListUpdate() callback as defined in clause 6.2.3.17.

When the getFdServices() call is received by the MBMS client as defined in clause 6.2.3.4, the MBMS client sets the parameters as follows:

- If the _service[] list is empty, then an empty list is returned.
- For each MBMS user service in the service[] list, one service record is generated as follows:
  - The value of the attribute _serviceId is assigned to serviceId.
  - The value of the attribute _serviceClass is assigned to serviceClass.
  - The value of _serviceLanguage is assigned to serviceLanguage.
  - For each record in the _serviceName[] one serviceNameList entry is generated and:
    - the name is set to the value _name,
    - the language is set to the value _language.
  - The value of _serviceBroadcastAvailability is assigned to serviceBroadcastAvailability.
  - If at least one _sessionSchedule[] record is present then:
    - The activeDownloadPeriodStartTime is set to the value of earliest _start time of any entry in the _sessionSchedule[]such that the corresponding _stop time is in the future.
    - The activeDownloadPeriodStopTime is set to the value of the _stop time of the entry selected earliest start time.
  - If no _sessionSchedule[] record is present, or the _start and _stop on existing entries in _sessionSchedule[] are in the past:
    - The activeDownloadPeriodStartTime is set to 0.
    - The activeDownloadPeriodStopTime is set to 0.

When the setFdStorageLocation() method as defined in sub-clause 6.2.3.5 is received by the MBMS client, the MBMS client runs the following steps:

- It updates the internal variable _locationPath to the parameter value of locationPath provided in the call.

NOTE: The MBMS client provides any new files for that MAA at the locationPath, but not ongoing file capture.

- If the storage location is empty the MBMS client selects a local directory on the device, which the MAA can access directly via file access interfaces or via HTTP GET methods (see sub-clauses 6.2.2.5 and 6.2.2.6).

When the setFdServiceClassFilter() is received, the MBMS client runs the following steps:

- It replaces the internal variable _serviceClass[] with the parameter values provided in serviceClassList.

  Note that this does not necessarily change the ongoing MBMS client operation, e.g. capturing.

- The MBMS client issues a fdServiceListUpdate() notification as defined in clause 6.2.3.17 to the MAA to notify it of this effect.

When the startFdCapture() method is received, the MBMS client runs the following steps:

- The MBMS client checks for errors and if necessary, the fdServiceError() notification as defined in clause 6.2.3.18 is initiated.
- If the requested fileURI matches an existing outstanding startFdCapture() request as recorded on the _fileCaptureRequest[], the internal error code is set to FD_DUPLICATE_FILE_URI and the fdServiceError() notification as defined in clause 6.2.3.18 is initiated.
- If the requested fileURI is ambiguous in the following manner
  - The fileURI is an absolute URL or a Base URL and there is an existing outstanding startFdCapture() with an empty fileURI, or
  - fileURI is an absolute URL and there is an existing outstanding startFdCapture() with an Base URL in the fileURI that is base URL for the aboluteURL.

  then, the internal error code is set to FD_AMBIGUOUS_FILE_URI and the fdServiceError() notification as defined in 6.2.3.18 is initiated.
- Otherwise, The fileURI is added to the internal list of the MBMS client _fileCaptureRequest[]. Any overlapping entry should be avoided.
- The MBMS client removes existing outstanding startFdCapture() requests from _fileCaptureRequest[] when the requested fileURI on a startFdCapture() is broader (i.e., superseding older requests) than these existing outstanding startFdCapture() requests; this request consolidation will not impact ongoing file downloads, specifically:
  - When fileURI is empty on the new startFdCapture(), all existing outstanding startFdCapture() are removed.
  - When fileURI is a Base URL, existing outstanding startFdCapture() requests with an absolute URL are removed if the new fileURI in the request is a base URL for the absolute URL on these existing outstanding startFdCapture().
- The _disableFileCopy is set to the value of disableFileCopy.
- The _captureOnce is set to the value of captureOnce.
- The MBMS client is in CAPTURE_NOTIFY mode.

Whenever there has been a change to the parameters reported to the MAA in response to a getFdServices() API, i.e. in the internal service class list _serviceClass[] to add a new service record to the list or a change in one of the following internal parameters in the service record in the _serviceLanguage, _serviceName[]_serviceBroadcastAvailability, or updates to the _fileURIStatus[] adding new _URIs with _deliveryState set to FD_SCHEDULED, the MBMS client notifies MAA with fdServiceListUpdate() as defined in clause 6.2.3.17.

When the deregisterFdApp() is received, the client moves to IDLE state.

6.2.2.5 MBMS Client Operation in CAPTURE_NOTIFY State

In the CAPTURE_NOTIFY state, the MBMS client carries out all actions as in the REGISTERED state.

For one MBMS user service identified with one service record in the internal parameter `_service[]` with a specific `_serviceID` the MBMS client continuously updates the internal parameters for this service.

In addition, for each `fileCaptureRequest[]` record in the service record:

1) If the `_fileUri` is empty then the MBMS client receives all new and updated files delivered on this MBMS service with service ID `serviceId`. Updated files may be discovered by a change of the MD5 in the FDT.

2) If the `_fileUri` is a complete absolute URI, then the MBMS client receives only that file delivered on the MBMS service with service ID `serviceId` which has a matching URL as the `fileURI` parameter if the file is received for the first time or is a new version since the last reception.

3) If the `_fileUri` is a Base URI as defined in RFC 3986 [11], then the MBMS client receives all new and updated files delivered on the MBMS service with service ID `serviceId` which are delivered through the MBMS user service with a matching BaseURL of the one defined in the `fileURI` parameter.

Furthermore, the MBMS Client performs the actions as follows.

4) For each file announced via the file schedule element and matching any of the capture requests in `_fileCaptureRequest[]`:

   - the MBMS client creates an entry in the `_fileURIStatus[]` adding the `_URI` and delivery schedules `_deliverySchedule[]` and sets the `_deliveryState` to `FD_SCHEDULED`.

   - The MBMS Client should send a `fdServiceListUpdate()` callback as defined in clause 6.2.3.17.

5) For each file announced in the FDT and matching any of the capture requests in `fileCaptureRequest[]`, the MBMS client:

   - Updates or creates an entry adding the `_URI.`

   - Sets the `_deliveryState` to `FD_IN_PROGRESS.`

   - Does not download the same version of the file if the `Content-MD5` in the **`File`** element of the FDT Instance is the same as a previously received version of the file.

   - If a MIME type was defined via the FDT describing that file transmission, the `_contentType` parameter is set to the value of the `Content-Type` as in the **`File`** entry of the FDT. If the MIME type is not defined, the `_contentType` parameter is set to an empty string. The MD5 may be extracted from the FDT if present.

6) For each successfully received file announced in the FDT and matching any of the capture requests in `fileCaptureRequest[]`:

   - The MBMS client updates the `_deliveryState` to `FD_RECEIVED`.

   - If the `_locationPath` is defined and if the MBMS client is successful in copying/moving the collected file to the directory in `_locationPath`, the MBMS client sets the `_appState` to moved and the `_fileLocation` pointing to the file in the `_locationPath`.

   - If the `_locationPath` is empty, (i.e. `_appState` indicates that MBMS client space needs to be used or that the MBMS client selected a location on the MAA space) or if the MBMS client is not successful in copying/moving the collected file to the directory defined in `_locationPath` (`_appState` is set to indicate the failure) then the `_fileLocation` is set to:

     - A complete file name (including the directory path) on the UE local file system where the file can be accessed by the MAA. The file may be stored under an MBMS client defined directory that is accessible to the MAA.

     - An HTTP URL where the MAA can retrieve the file using the HTTP GET method. This format may be used when the file is stored on a location that is not directly accessible to the MAA.

   - The MBMS client announces it through `fileAvailable()` notification as defined in clause 6.2.3.8 to the MAA by setting the parameters as follows:

- The `serviceId` is set to the `_serviceID`.
- The `fileUri` is set to the value of `_URI`.
- The `fileLocation` is set to the value of the `_fileLocation`.
- The `contentType` is set to the `_contentType`.
- If the `_appState` is failed or internal, the MBMS client sets the value of the `availabilityDeadline` to the internal variable `_defaultAvailabilityDeadline`, otherwise sets the value to 0.
- The internal `_notified` flag is set to true.

- If the `_captureOnce` is set to true, the corresponding `fileURI` is excluded/removed from the `_fileCaptureRequest[]` and from the internal `_fileURIStatus[]` list.

7) For each non-successfully received file announced in the FDT and matching any of the capture requests in `fileCaptureRequest[]` (after file repair failed or when file repair is not defined for the service):

- The MBMS client sets the sets the `_deliveryState` to `FD_SCHEDULED`
- The MBMS client announces it through `fileDownloadFailure()` notification as defined in 6.2.3.10 to the MAA with the following parameter settings:
  - The `serviceId` is set to the `_serviceID`.
  - The `fileUri` is set to the value of `_URI`.

Whenever the in the internal `_fileURIStatus[]` was changed, the MBMS client should issue a `fileDownloadStateUpdate()`notification as defined in 6.2.3.15 with the following parameters:

- The `serviceId` is set to the `_serviceID`.

The MBMS client moves to `REGISTERED` state if the `_fileCaptureRequest[]` is empty.

When the `getFdAvailableFileList()` API call as defined in clause 6.2.3.12 is received, the MBMS client runs the following steps:

- For the service with internal `_serviceId` set to the input parameter `serviceID`, for each successfully received file (i.e. internal `_deliveryState` being received) that matches a `_fileCaptureRequest[]` and is included in the `_fileURIStatus[]` record with an internal status `_notified` set to false, the following parameters are assigned
  - The `fileUri` is set to the value of `_URI`.
  - The `fileLocation` is set to the value of the `_fileLocation`.
  - The `contentType` is set to the `_contentType`.
  - If the `_appState` is failed or internal, the MBMS client sets the value of the `availabilityDeadline` to the internal variable `_defaultAvailabilityDeadline`, otherwise sets the value to 0.
  - The internal status `_notified` set to true.

When the MBMS client receives a `stopFdCapture()` request as defined in clause 6.2.3.13 that matches an entry in the internal parameter `_fileCaptureRequest[]`:

- The MBMS client removes this entry from the `_fileCaptureRequest[]` record in order to stop any on-going and future file receptions that match that particular record.
- The MBMS client should keep records of outstanding `startFdCapture()` requests that are unambiguous such that a `stopFdCapture()` can unambiguously remove an entry in the internal parameter `_fileCaptureRequest[]` from an earlier `startFdCapture()` request.

- The MBMS client may also send a failure indication via the `fdServiceError()` with the `FD_AMBIGUOUS_FILE_URI` error code when the requested fileURI on a `stopFdCapture()` is more specific than an existing outstanding `startFdCapture()` requests that is broader:
  - When `fileURI` is an absolute URL or a Base URL and there exists an entry in the internal parameter `_fileCaptureRequest[]` with an empty fileURI.
  - When fileURI is an absolute URL and there exists an entry in the internal parameter `_fileCaptureRequest[]` with an empty fileURI. With an Base URL in the `_fileURI` that is base URL for the aboluteURL.
- The MBMS client may send a failure indication via the `fdServiceError()` with the `FD_STOP_FILE_URI_NOT_FOUND` error code to any `stopFdCapture()` request that does not match an entry in the internal parameter `_fileCaptureRequest[]`.

When the `getFdActiveServices()` API call as defined in clause 6.2.3.14 is received, the MBMS client runs the following steps:

- For the service with internal `_serviceId` set to the input parameter `serviceID`, for each entry in the internal parameter `_fileCaptureRequest[]` one record in the response is set with the `fileUri` set to the value of `_URI`.

When the MBMS Client receives `getFdDownloadStateList()`API call as defined in clause 6.2.3.16, it runs the following steps:

- For the service with internal `_serviceId` set to the input parameter `serviceID`, for each file that matches a `_fileCaptureRequest[]` and is included in the `_fileURIStatus[]` record, the following parameters are assigned:
  - The `fileUri` is set to the value of `_URI`.
  - The `state` is set to the internal value `_deliveryState`.

If the `deregisterFdApp()` API call invoked the MBMS client:

- And if the `_fileCaptureRequest[]` record contains no more entries or the `_registrationValidityDuration` is set to zero, the MBMS client moves to `IDLE` state.
- If the `_fileCaptureRequest[]` record contains one or more entries, the MBMS client moves to `CAPTURE_BACKGROUND` state.

6.2.2.6 MBMS Client Operation in CAPTURE_BACKGROUND State

When the MBMS client enters this state, the internal `_backgroundCaptureTimer` is set to `_registrationValidityDuration` and started. The MBMS client remains in this state until this timer expires, or the MAA invokes a new `registerFdApp()` API call.

Once the time for the registration validation has expired, the MBMS client clears all context for the MAA and returns to `IDLE` state for the MAA. Until this is the case, the MBMS client performs the following actions.

The MBMS client carries out all actions as in the `CAPTURE_NOTIFY` state, except for those associated with sending notifications (invoking call-back functions) to the app.

While an MAA is deregistered and files are received for that MAA, if multiple versions of the same file (i.e., the same `fileURI` but different `Content-MD5` in the FDT for a File Delivery Application Service) are received, only the last file version received is kept by the MBMS client and made available to the MAA after the new registration.

If the MAA is not currently registered at the time and the download of files for that MAA fails, the MBMS client is not able to inform MAA.

For one MBMS user service identified with one service record in the internal parameter `_service[]` with a specific `_serviceID` the MBMS client continuously updates the internal parameters for this service. Whenever the delivery state is changed in the state, the internal `_notified` flag is set to false.

6.2.3 Methods

6.2.3.1 Overview

Table 6.2.3.1-1 provides an overview over the methods defined for the File Delivery Application Service API. Different types are indicated, namely state changes triggered by the MAA, status query of the MAA to the client, parameter updates as well as notifications from the client. The direction of the main communication flow between the MAA (A) and the MBMS client (C) is provided.

Table 6.2.3.1-1: Methods defined for File Delivery Application Service API

Method	Type	Direction	Brief Description	Clause
`registerFdApp`	State change	A -> C	MAA registers a callback listener with the MBMS client	6.2.3.2
`deregisterFdApp`	State change	A -> C	MAA deregisters with the MBMS client	6.2.3.9
`startFdCapture`	State change	A -> C	Start download of files over file delivery service	6.2.3.7
`stopFdCapture`	State change	A -> C	Stop download of files for the file delivery service	6.2.3.13
`getFdActiveServices`	Status query	C <-> A	Get list of currently active services	6.2.3.14
`getFdAvailableFileList`	Status query	C <-> A	Retrieves the list of files previously captured for the MAA	6.2.3.12
`getFdServices`	Status query	C <-> A	Retrieves the list of File Delivery services defined in the USD	6.2.3.4
`getFdDownloadStateList`	Status query	C <-> A	Retrieves the state of files pending download	6.2.3.16
`getVersion`	Status query	C <-> A	the version of the File Delivery Application Service interface	6.2.3.21
`setFdServiceClassFilter`	Update to parameter list	A -> C	MAA sets a filter on file delivery services in which it is interested	6.2.3.6
`setFdStorageLocation`	Update to parameter list	A -> C	Sets the storage location to store the MAA downloaded files	6.2.3.5
`registerFdResponse`	Response	C -> A	The response to the MAA service register API	6.2.3.3
`fileAvailable`	Notification	C -> A	Notification to MAA when a new file is downloaded per MAA capture request	6.2.3.8
`fdServiceListUpdate`	Notification	C -> A	Notification to MAA on an update of the available for file delivery services	6.2.3.17
`fdServiceError`	Notification	C -> A	Notification to MAA when there is an error with broadcast download of service	6.2.3.18
`fileDownloadFailure`	Notification	C -> A	Notification to MAA that download of a requested file failed	6.2.3.10
`inaccessibleLocation`	Notification	C -> A	Notification to MAA that the storage location set by the MAA is not accessible by the MBMS Client.	6.2.3.20
`insufficientStorage`	Notification	C -> A	Notification to MAA indicating a warning on the low storage condition	6.2.3.19
`fileDownloadStateUpdate`	Notification	C -> A	Notify MAA of a change in the state of pending file downloads	6.2.3.15
`fileListAvailable`	Notification	C -> A	Notify MAA when the list of downloaded files is available to retrieve	6.2.3.11

6.2.3.2 Registration

6.2.3.2.1 Overview

This clause defines `registerFdApp()` interface.

An MAA calls the `registerFdApp()` interface to register with the MBMS Client to consume File Delivery Application Services. The `registerFdApp()` interface has two purposes:

1) It signals to the MBMS Client that an MAA is interested to consume MBMS services. This registration may be considered as pre-condition for the MBMS Client to keep checking for updates to the File Delivery Application Services defined.

2) It allows the MAA to identify its callback listeners defined in the File Delivery Application Service API for the MBMS Client to provide asynchronous notifications to the MAA on relevant events associated with the reception of files.

NOTE: Since some application development frameworks do not support callback functions, an MAA for these frameworks will not provide callback listeners in the `registerFdApp()` interface. Instead, the MAA will implement the necessary approach available on these frameworks to receive event notifications from the MBMS Client in place of callback functions. The notifications implemented on these frameworks will include the same information content as defined on the structures for the IDL callback functions.

Figure 6.2.3.2.1-1 shows a call flow and the usage of the `registerFdApp()` interface.

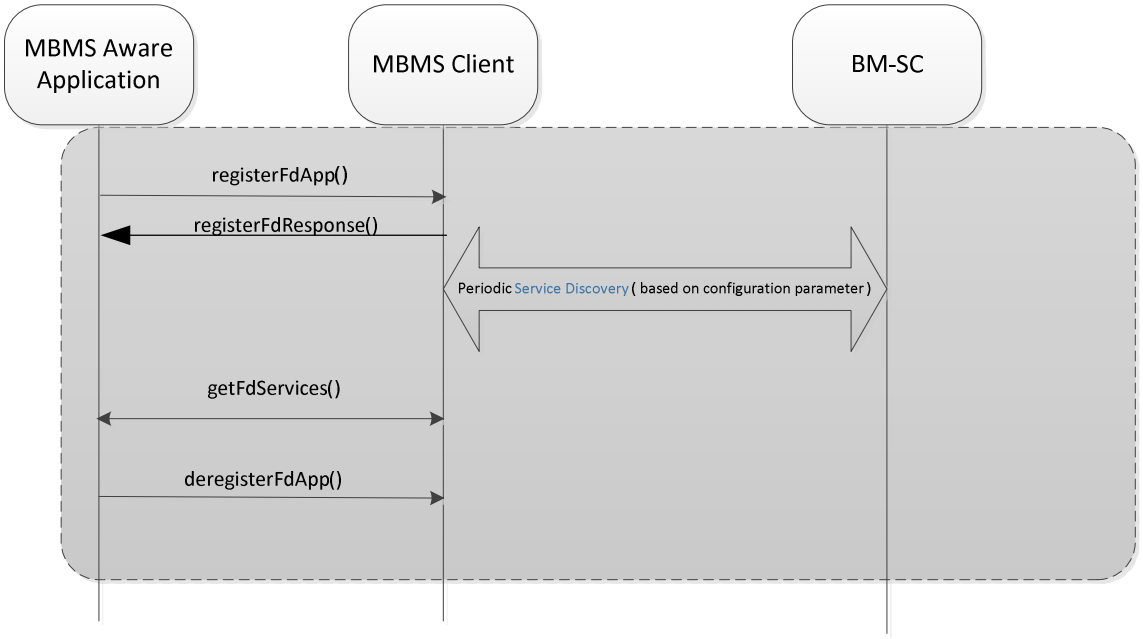

Figure 6.2.3.2.1-1: MAA Registration sequence diagram

6.2.3.2.2 Parameters

The parameters for the `registerFdApp()` API are:

- **`string`** `appId` – provides a unique ID for the application registering with the MBMS client, which uses this identity to maintain state information (e.g., capture requests and captured files) for a particular MBMS Aware Application. The uniqueness of the ID is in the context of any application that may possibly register with MBMS client. Uniqueness is typically provided at the platform level.

- **`any`** `platformSpecificAppContext` – a platform-specific context for the registering application that enables the MBMS client to get extra information about the application that may be need to enable the application to have access to MBMS services, e.g., to enable application authentication or to enable the application to communicate with the MBMS client via platform (e.g., HLOS) services.

- **sequence<string>** serviceClassList – provides a comma-separated list of service classes which the application is interested to register. Each service class string can be any string or it may be empty.
- **StorageLocation** locationPath – identifies a local directory available on the device storage, which the application can access and where successfully collected files can be copied/moved before notifying that the file is available to the application. The storage location is a string pointing to a directory, or it may be empty, if no location is provided.
- **unsigned long** registrationValidityDuration – the period of time in seconds following the application de-registration (via the application call to deregisterFdApp(), or when the MBMS client detects that the application is no longer running), and possible exit, over which the eMBMS client still considers the application registered for the purpose of fulfilling any outstanding startFileCapture() requests. The default value of this option is 0 (zero) which signals to the MBMS client to clear any outstanding startFdCapture() requests for that application upon its de-registration, and possible exit.

NOTE 1: The registrationValidityDuration is provided via the registerFdApp(), rather than via a deregisterFdApp(), to handle abnormal application termination conditions (e.g., application crash or it is terminated by the HLOS) before the application has a chance to invoke the deregisterFdApp().

- **ILTEFileDeliveryServiceCallback** callBack – provides the MBMS client with the call back functions associated with File Delivery Application Service APIs for the registering MBMS Aware Application.

NOTE 2: The callback element in the IDL description is optional and only included when the application development framework supports programmatic callback interfaces. If callbacks are not supported on a given application development framework, the same information content as defined on the callback structures is to be provided to the application via the notification method available with that development framework when the respective condition is met.

6.2.3.2.3 Pre-Conditions

The application is assigned a unique application ID appId in the context of its operation (e.g., a smartphone HLOS) with the MBMS client.

The application is pre-configured with the set of service classes that allows it to consume the File Delivery Application Services associated with these service classes.

The application manages the storage of requested files, especially the amount of storage to be used.

The application may use this method at launch or after a deregisterFdApp() has been called.

The application assumes the client in IDLE state or CAPTURE_BACKGROUND state.

The MBMS client may be configured to support a maximum _maxRegistrationValidityDuration.

6.2.3.2.4 Usage of Method for Application

The application uses the method registerFdApp() to register with the MBMS Client to consume File Delivery Application Services.

The application provides its appId and, if applicable, some platform specific application context, platformSpecificAppContext.

The application provides the set of service classes which the application is interested to register.

Setting the registrationValidityDuration enables the application to allow the MBMS client to capture files in the background when the application is not currently running; typically after invoking the deregisterFdApp()or after crashing. The application developer should be aware that received files belong to the application and that the MBMS client does not provide content management functions beyond reception and *temporary* storage of received files in between consecutive runs of the application.

When the application is not interested in receiving files from File Delivery Application Services while the application is not running, the application should set the registrationValidityDuration to 0 (zero). When the application is interested in receiving files from File Delivery Application Services while the application is not running, the

application should set the `registrationValidityDuration` to a non-zero value and it should call the `deregisterFdApp()` interface before stop running to disable notifications from the File Delivery Services APIs. If the MBMS client can determine that an application has crashed and exited without invoking the `deregisterFdApp()` interface, the MBMS client will assume that the application had called the `deregisterFdApp()` before exiting.

On issuing a `registerFdApp()` following a `deregisterFdApp()`, the application will include the same `serviceClassList` used on the last `registerFdApp()` or `setFdServiceClassFilter()`to signal interest on the same set of service classes identified before. The application may change the list of registered service classes by providing a `serviceClassList` as part of a new `registerFdApp()` request, similarly to using the `setFdServiceClassFilter()` method.

The application should define a local directory `locationPath` on the device, where successfully collected files will be copied/moved before making the file available to the application. If no directory is defined, the `locationPath` should be an empty string.

The application selects a value for the parameter `registrationValidityDuration` to constrain the amount of storage the MBMS client will use to collect and keep application-requested files in the MBMS client storage space while the application is not currently running (e.g., has de-registered or crashed). The amount of files cached for an application that does not collect its files may impact the MBMS client's ability to collect files for other applications. In selecting a value for the registrationValidityDuration the following should consider:

- The frequency and amount of files received during the selected registrationValidityDuration. For instance, assuming that 100MB worth of files are delivered every day:
  - The application could select a registrationValidityDuration of Nx24h (e.g., N=10) to request the MBMS client to store at most Nx100MB (e.g., 1GB) if the application is not run by the user in Nx24h.
  - The application could ask its user (or have a preconfigured behavior on) how long the user wants files to be collected in-between the user's access to files delivered to the application. If the user selects a long period (e.g., 2months) the application should not use that large values as the registrationValidityDuration (e.g., this could mean 6GB in the example above). Instead, the application should include behaviors to periodically re-register (e.g., every 5days) and collect received files to manage storage of its application files. Leaving those files in the MBMS client storage space (e.g., 6GB) could exceed the MBMS client storage space allowance and impact the reception of files for other applications.
- The relevance of older vs. newer files when managing the storage for files received if the user does not access the application over a long period of time.
  - The application could ask the user how much storage to use for received files and whether to delete older files (newer files preferred), or stop new downloads (older preferred), or however else the application choses to support managing received files.
  - As described above, in the absence of the user launching the application for File Delivery Application Services with outstanding `startFdCapture()` requests, that application should automatically re-register (e.g., every 5days as discussed above) with the MBMS client with a periodicity not greater than registrationValidityDuration and retrieve files captured during the period the application was not currently registered.
  - The application should then manage the downloaded files with respect to the amount of storage consumed by files of that application. For instance, the application may prioritize retaining newer versus older files or let the registrationValidityDuration expire (therefore causing the MBMS client to stop continued file downloads for that application) if the user does not consume file contents for that application.

EXAMPLE 1: Daily headline news application allows the user to collect files from two File Delivery Application Services with new/updated video clip files downloaded twice every day.

- The application registrationValidityDuration is 2days, it re-registers with the MBMS client every 1.5 days and it keeps only the files that are no more than two days old, e.g., as configured by the user.

EXAMPLE 2: Weekly magazine application allows the user to collect files from a File Delivery Application Service with new/updated files for selected electronic versions of weekly magazines downloaded once a week.

- The application registrationValidityDuration is 15 days, it re-registers with the MBMS client every 7 days and it keeps only the files that are no more than three weeks old, e.g., as configured by the user.

6.2.3.2.5 Expected MBMS Client Actions

When this method is received, the MBMS client registers the app. For more details refer to clause 6.2.2.3.

6.2.3.2.6 Post-Conditions

The application expects a `registerFdResponse()` as defined in clause 6.2.3.3.

6.2.3.3 File Delivery Application Service Registration Response

6.2.3.3.1 Overview

This subclause defines the `registerFdResponse()` call.

As illustrated in Figure 6.2.3.2.1-1, the MBMS client responds to an MAA call to the `registerFdApp()` API with a `registerFdResponse()` call back providing the result of the registration request.

6.2.3.3.2 Parameters

The parameters for the `registerFdResponse()` API are:

- `EmbmsCommonTypes::RegResponseCode` value – provides a result code on the registration request. The allowed values are:
  - `REGISTER_SUCCESS` – indicates that the registration has been processed successfully and the application can proceed with other API interactions with the MBMS client for File Delivery Application Services.
  - `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE` – Indicates that the registration has failed since the File Delivery Application Service API did not find an MBMS client available on the UE on which the application is running and no MBMS service will be available to the application.
  - `MISSING_PARAMETER` – indicates that the registration has failed since one or more of the required parameter was missing.
- `string message` – provides an associated text description of the error message. The message may be empty.
- `unsigned long acceptedFdRegistrationValidityDuration` – when returning `REGISTER_SUCCESS`, this parameter indicates the registration validity duration the MBMS client will provide to the registering application.

6.2.3.3.3 Pre-Conditions

The MBMS client has received a call via the `registerFdApp()` API with the parameters documented in clause 6.2.3.2.2.

6.2.3.3.4 Expected MBMS Client Actions

Based on the parameters of the `registerFdApp()`, the MBMS client provides the response and changes the state. For more details see clause 6.2.2.3.

6.2.3.3.5 Usage of Method for Application

Once the application receives a the `registerFdResponse()` with the `RegResponseCode` set to `REGISTER_SUCCESS`, the application can proceed with other API interactions with the MBMS client, for instance, by issuing `startFdCapture()` requests.

Based on the response in `acceptedFdRegistrationValidityDuration` the application should adjust its expectations accordingly if the value returned is not what was requested.

If the MBMS client is temporarily in NOT_AVAILABLE, if the registerFdResponse() signaled a failure with a FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE, the application may periodically recheck if the state of the MBMS client changes by retrying the registerFdRequest() API.

6.2.3.3.6 Post-Conditions

If the MBMS client functions cannot be activated and once the response is sent with a FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE, then MBMS client is at least temporarily in NON_AVAILABLE state.

If the MBMS client functions cannot be activated and once the response is sent with a MISSING_PARAMETER, then MBMS client is at least temporarily in IDLE state.

If the MBMS client functions can be activated and a response is sent with REGISTER_SUCCESS, then MBMS client is in REGISTERED state with the REGISTERED state parameters as set above.

6.2.3.4 Getting information on available File Delivery Application Services

6.2.3.4.1 Overview

This clause defines the getFdServices() interface.

The getFdServices() interface returns the complete list of available File Delivery Application Services information. After a successful registration with the MBMS client, the MBMS Aware Application can use the getFdServices() API to discover the available File Delivery Application Services associated with the service classes registered via the registerFdApp(). Also the MBMS client may use the getFdServices() API after receiving a fdServiceListUpdate() call back notification to changes to the available File Delivery Application Services.

6.2.3.4.2 Parameters

The getFdServices() API does not have any input parameters.

The getFdServices() API returns a list describing the available File Delivery Application Service, where each service is described by the following output only parameters:

- sequence<ServiceNameLang> serviceNameList – optionally provides a list of the service title name in possibly different languages. Each (name, lang) pair defines a title for the service on the language indicated.
  - string name – offers a title for the user service on the language identified in the lang parameter.
  - string lang – identifies a natural language identifier per IETF RFC 3066 [10].
- string serviceClass – identifies the service class which is associated with the service.
- string serviceId – provides the unique service ID for the service. The uniqueness is among all services provided by the BMSC.
- string serviceLanguage – indicates the available language for the service and represented as an identifier per IETF RFC 3066 [10].
- EmbmsCommonTypes::ServiceAvailabilityType serviceBroadcastAvailability – signals whether the UE is currently in the broadcast coverage area for the service.
  - The possible values are for a File Delivery Application Service:
    - -BROADCAST_AVAILABLE – if content for the service is broadcast at the current device location.
    - BROADCAST_UNAVAILABLE – if content for the service is not broadcast at the current device location.
- sequence<string> fileUriList – optionally provides a list of file names for the files that are currently scheduled to be transmitted.

- `EmbmsCommonTypes::Date activeDownloadPeriodStartTime` – signals the current/next active File Delivery Application Service start time, when files start being broadcast over the air.
- `EmbmsCommonTypes::Date activeDownloadPeriodEndTime` – signals the current/next active File Delivery Application Service stop time, when files stop being broadcast over the air.

Note that the available File Delivery Application Service list may be empty.

6.2.3.4.3 Pre-Conditions

The MBMS client is in `REGISTERED` state or in `CAPTURE_NOTIFY` state and may or may not have acquired any USD information for services that are included in the service class list.

The MBMS client may have updated internal `_service[]` list.

6.2.3.4.4 Expected MBMS Client Actions

When this method is received, the MBMS client returns its internal parameters. For more details see clause 6.2.2.3.

6.2.3.4.5 Usage of Method for Application

The usage of the parameters `serviceNameList`, `serviceClass`, `serviceBroadcastAvailability,` and `serviceLanguage` is typically up to the application.

The `fileUriList` may be used by the application for selective file reception, e.g. when used together with `startFdCapture()` as defined in clause 6.2.3.7.

The application should use this call right after the `registerFdResponse()` notification as defined in clause 6.2.3.3 is received or after the `fdServiceListUpdate()` notification as defined in 6.2.3.17 is received.

The application shall use the `serviceId` to identify the service in subsequent communication with the MBMS client to manage the files to be captured on that service for the application.

The parameters `activeDownloadPeriodStartTime` and `activeDownloadPeriodEndTime` provides the application the ability to determine the current broadcast state for the service as follows:

- If the current time is such that `activeDownloadPeriodStartTime` ≤ current time ≤ `activeDownloadPeriodEndTime`, files may be captured for the service at the current time.
- If the `activeDownloadPeriodStartTime` is in the future, there is currently file capturing expected for the service, but delivery is currently scheduled to start at this advertised time.
- If the `activeDownloadPeriodStartTime` is set to zero, there is no currently defined broadcast schedule time for the service.
- If the `activeDownloadPeriodEndTime` is in the past, or there are no `activeDownloadPeriodStartTime` and no `activeDownloadPeriodEndTime` defined for the service, then there is currently no broadcast being made for the service, and there is no currently scheduled broadcast time for the service.

6.2.3.4.6 Post-Conditions

This call does not change the MBMS client state.

6.2.3.5 Establishing the location where files are stored for an application

6.2.3.5.1 Overview

This clause defines `setFdStorageLocation()`.

While an application is actively registered with the MBMS client to consume File Delivery Application Services, the MBMS Aware Application can call the `setFdStorageLocation()` API to set or update the location where files collected for the application are to be stored. This message flow is shown in Figure 6.2.3.5.1-1.

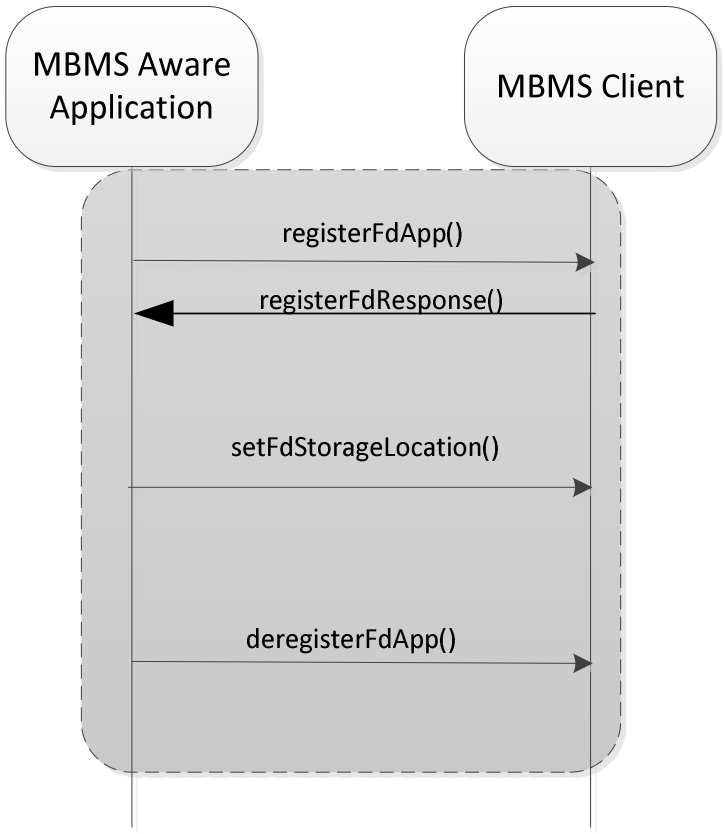

Figure 6.2.3.5.1-1: Sequence diagram for updating the storage location for collected files

6.2.3.5.2 Parameters

The parameters for the `setFdStorageLocation()` method are:

- StorageLocation locationPath – see clause 6.2.3.2.2.

6.2.3.5.3 Pre-Conditions

The application is registered with the MBMS client to consume File Delivery Application Services, and MBMS client is in `REGISTERED` or `CAPTURE_NOTIFY` state.

6.2.3.5.4 Usage of Method for Application

The MBMS Aware Application may invoke the `setFdStorageLocation()` API call to update the previously defined storage location. This includes the case that the `storageLocation` was not set previously.

It may also use this API call if it receives an insufficient storage notification as defined in clause 6.2.3.19.

6.2.3.5.5 Expected MBMS Client behaviour

When this method is received, the MBMS client updates the internal parameters and notifies the client. For more details see clause 6.2.2.3.

6.2.3.5.6 Post-Conditions

The MBMS client internal variable `_locationPath` for this application is updated. No state change is applied.

6.2.3.6 Updating the registered service classes

6.2.3.6.1 Overview

This clause defines `setFdServiceClassFilter()`.

While an application is actively registered with the MBMS client to consume File Delivery Application Services, the MBMS Aware Application can call the `setFdServiceClassFilter()` API to update the list of service classes the application wants to be registered with, see Figure 6.2.3.6.2-1.

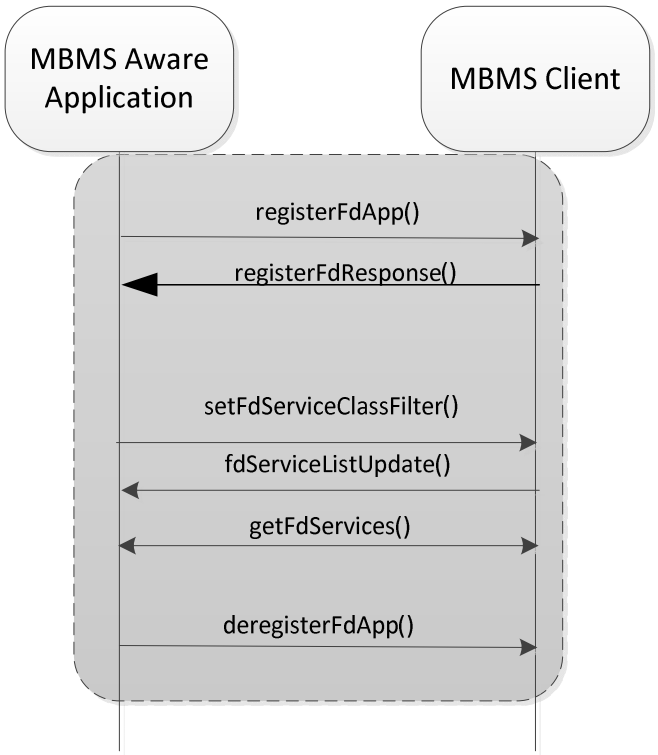

Figure 6.2.3.6.2-1: Sequence diagram for updating the registered service classes for an application

6.2.3.6.2 Parameters

The parameters for the `setFdServiceClassFilter()` method are:

- sequence<string> serviceClassList – see clause 6.2.3.2.2.

6.2.3.6.3 Pre-Conditions

The application is actively registered with the MBMS client to consume File Delivery Application Services, and MBMS client is in `REGISTERED` state or `CAPTURE_NOTIFY` for the application.

6.2.3.6.4 Usage of Method for Application

The MBMS Aware Application may invoke the `setFdServiceClassFilter()` API to update the previously defined list of service classes that includes additional service classes or includes fewer service classes than the list of service classes.

The application should be aware that the updates are only active once an `fdServiceListUpdate()` notification is received that confirms the new service class filters.

6.2.3.6.5 Expected MBMS Client Actions

When this method is invoked, the MBMS client updates the internal parameters and notifies the client.

6.2.3.6.6 Post-Conditions

The MBMS client issues an `fdServiceListUpdate()` notification as defined in clause 6.2.3.17. No state change is applied.

6.2.3.7 Start File Delivery Capture

6.2.3.7.1 Overview

This clause defines the `startFdCapture()` API.

Once the File Delivery Application Service registration is complete, the MBMS Aware Application can make calls on the `startFdCapture()` API to selected `fileURI`s for the files to be received through the MBMS service as shown in Figure 6.2.3.7.1-1.

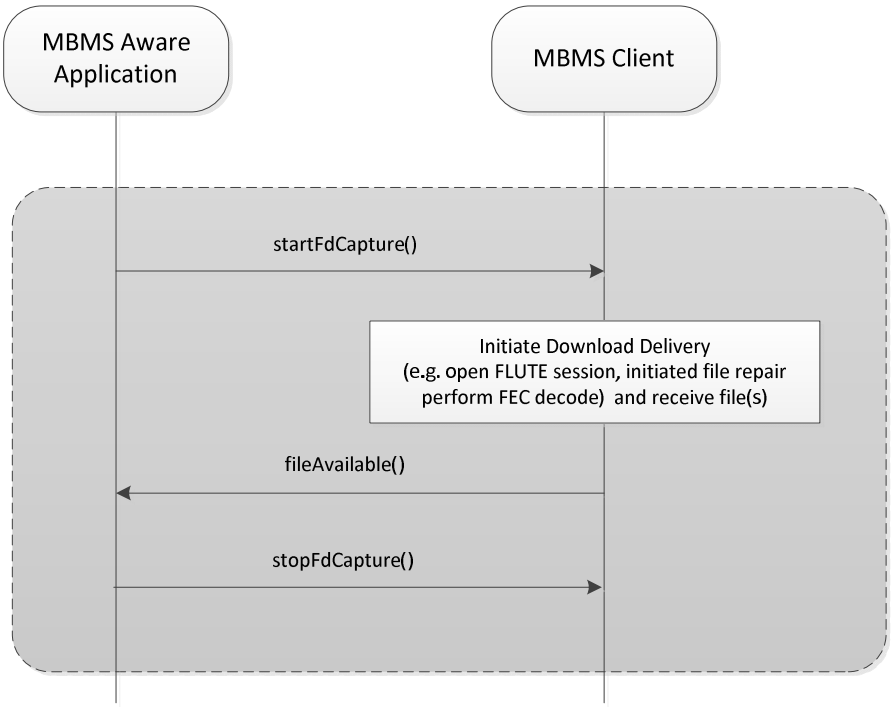

Figure 6.2.3.7.1-1: File Delivery Application Start Capture

6.2.3.7.2 Parameters

The parameters for the `startFdCapture()` API are:

- `string serviceId` – see clause 6.2.3.4.2. The service ID for the service for which the files are captured.
- `string fileUri` – identifies the files to be captured on the service identified in `serviceId`. Allowed values include:
  - The empty string signals that the MAA is interested in receiving all new files and updates to previously received files.
  - A BaseURL, i.e., a complete path for subdirectory (a prefix) identifying a group of files under that directory.
  - An absoluteURL, i.e., a complete URL that identifies a single file resource.
- `boolean disableFileCopy` – when set to true, this signals that the MAA does not want the MBMS client to make the file available on the MAA space.
- `boolean captureOnce` – when set to true, signals that the file requested via the fileURI (or a file matching a BaseURL in fileURI, or any file if the fileURI is empty) is to be captured only once.

6.2.3.7.3 Pre-Conditions

The application is registered with the MBMS client to consume File Delivery Application Services.

The `fileURI` format is not validated by the MBMS client.

6.2.3.7.4 Usage of Method for Application

The MBMS Aware Application can make calls on the `startFdCapture()` API to the select file URIs for the files to be received through the MBMS service with service ID `serviceId`. It is recommended that `http://` or file system names `c:/user/`… be used as file URIs.

An application may set the `fileURI` parameter to different values:

- An empty string signals that the application is interested in receiving all new files and updates to previously received files.
- If the `fileURI` is set to a Base URL as defined in RFC 3986 [11], the application may organize its files in a structured way (in a directory) such as to allow the identification of a group of files. For instance, a headline news clips application may group files under a …\sports\, a …\politics\, etc. folder and allow the user to select what type of headline news of interest and therefore request the MBMS client to capture all the files under …\sports\ if the user is only interested in sports headline news.
- An absolute URL, i.e., a complete URL that identifies a single file resource. A software update application on a given OEM device model may be preconfigured with an absolute URL for the file name that identifies the software image for that device model. That application would use that absolute URL as the file URI when request that its software image be received on a File Delivery Application Service.

The application should avoid overlap requests, e.g. requesting a fileURI and at the same time the Base URL that matches the fileURI.

The application may use the `getFdActiveServices()` API as defined in clause 6.2.3.14 to get back in synch with `fileURI` in remaining outstanding `startFdCapture()` requests.

When application is no longer interested in capturing specific files, it should call the `stopFdCapture()` interface as defined in clause 6.2.3.13.

A software update application on a given OEM device model may be preconfigured with an absolute URL for the file name that identifies the software image for that device model. That application would use that absolute URL as the file URI when request that its software image be received on a File Delivery Application Service.

6.2.3.7.5 Expected MBMS Client Actions

When this method is received, the MBMS client updates the internal parameters. For more details see clause 6.2.2.4.

6.2.3.7.6 Post-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state for the requested `serviceID`.

The parameters of the MBMS client are updated.

After capturing the files requested on a `startFdCapture()` request, the MBMS Client prepares to send `fileAvailable()` notification as defined via the registered callback listener.

6.2.3.8 File Available Notification

6.2.3.8.1 Overview

This clause defines the `fileAvailable()` callback function.

As illustrated in Figure 6, once the MBMS client has successfully collected a file that matches an outstanding `startFdCapture()` request from an MBMS Aware Application, the MBMS client invokes the `fileAvailable()` callback function.

6.2.3.8.2 Parameters

The parameters for the `fileAvailable()` API are:

- `string serviceId` – definition see above.
- The following is the file information for the received file:
  - `string fileUri` – identifies the file captured on the service identified in serviceId.
  - `string fileLocation` – identifies the location where the MBMS Aware Application can find the collected file.
  - `string contentType` – indicates the MIME type for the file identified in the fileUri.
  - `unsigned long availabilityDeadline` – signals a deadline in seconds when the file stored at the fileLocation will be removed from the MBMS client storage location.

6.2.3.8.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state for the `serviceId`.

The MBMS client has successfully received a file that is included in a `_fileCaptureRequest[]` record. Reception may be through broadcast only or may include unicast repair.

The MBMS client internal variable `_locationPath` may be defined or may not be defined.

The MBMS client has pre-configured an internal parameter `defaultAvailabilityDeadline` defining the time a file that is kept in the MBMS client owned storage location.

6.2.3.8.4 Expected MBMS Client Actions

The MBMS client invokes the `fileAvailable()` notification for each successfully received file that matches a `_fileCaptureRequest[]` and is included in the `_fileURIStatus[]`record. For more details see clause 6.2.2.5.

6.2.3.8.5 Usage of Method for Application

Once the application receives the callback, the application may access the file that is announced by the parameters in the callback function. If the storage is not defined by the `locationPath` application should access the file before the announced `availabilityDeadline`.

An MBMS Aware Application can retrieve the file using the HTTP GET method if the `fileLocation` is an HTTP URL.

The MBMS Aware Application needs to arrange for a copy of the file to the application space if the application intends to have access to the file beyond an `availabilityDeadline` indicated. The file may no longer be accessible after this time

6.2.3.8.6 Post-Conditions

The file is available at the location defined by the `fileLocation` parameter.

6.2.3.9 File Delivery Application Service De-registration

6.2.3.9.1 Overview

This clause defines the `deregisterFdApp()` call.

An MAA registers services classes with the MBMS client to request the capture of files on File Delivery Application Services, but the MAA does not have to be currently registered while files are being captured as discussed earlier. The MAA may deregister using the `deregisterFdApp()` call.

6.2.3.9.2 Parameters

None.

6.2.3.9.3 Pre-Conditions

The MBMS client is in `REGISTERED` or in `CAPTURE_NOTIFY` state for this application.

6.2.3.9.4 Usage of Method for Application

MBMS Aware Application registered with the MBMS client via the `registerFdApp()` API should invoke the `deregisterFdApp()` before exiting.

If the application deregisters, it will no longer receive notifications from the MBMS client. However, the MBMS client still captures files in the background until the registrationValidityDuration expires.

Nevertheless, the absence of notifications may result in unreceived files, if the MBMS client observes problems, e.g. not being able to access the storage location or the MBMS client having insufficient storage. Hence, the application should continuously check if the storage is still sufficient and accessible, even if is deregistered.

6.2.3.9.5 Expected MBMS Client Actions

The MBMS client no longer sends notifications.

If open file capture requests exist, the MBMS client continues collection files in the background until the time for `_registrationValidityDuration` expires.

For more details see 6.2.2.4.

6.2.3.9.6 Post-Conditions

The MAA is no longer registered with the MBMS client to receive notifications.

The MBMS client is either in `IDLE` mode or in `CAPTURE_BACKGROUND` mode.

6.2.3.10 File Download Failure Notification

6.2.3.10.1 Overview

This clause defines the `fileDownloadFailure()`callback function.

As illustrated in Figure 6.2.3.10.1-1, once the MBMS client has attempted to collect symbols for a file (possibly even via the unicast file repair procedure), that match an outstanding `startFdCapture()` request from an MAA, the MBMS client may still not be able to recover the file. Once the MBMS client detects that it failed FEC decoding the file, the MBMS client invokes the `fileDownloadFailure()` callback function (which the application registered with the MBMS client.

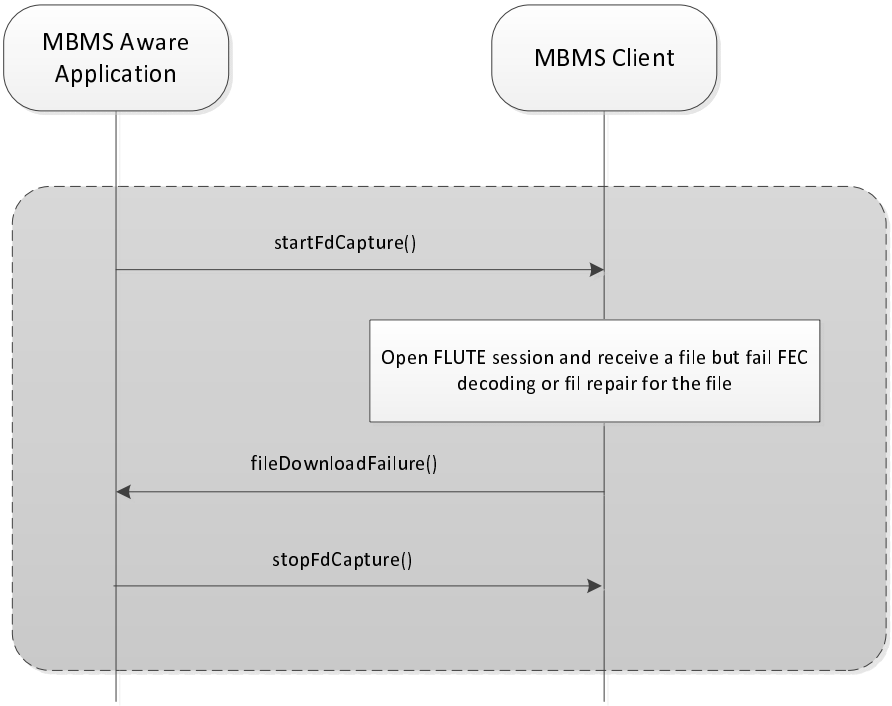

Figure 6.2.3.10.1-1: Signaling download failures

6.2.3.10.2 Parameters

The parameters for the `fileDownloadFailure()` API are:

- `string serviceId` – identifies the File Delivery Application Service on.
- `string fileUri` – identifies the file which failed being received.

6.2.3.10.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state for the `serviceId`.

6.2.3.10.4 Expected MBMS Client Actions

The MBMS client provides the `fileAvailable()` function for each non-successfully received file that matches a `_fileCaptureRequest[]` and is included in the `_fileURIStatus[]` record.

6.2.3.10.5 Usage of Method for Application

The application may use this information to identify other ways to access the file, or initiate actions to operate without this file.

6.2.3.10.6 Post-Conditions

The MBMS client may remain in `CAPTURE_NOTICE` state or if this was the last outstanding capture request it may move to REGISTERED state for the `serviceId`.

6.2.3.11 File List Available Notification

6.2.3.11.1 Overview

This clause defines the `fileListAvailable()`callback function.

As illustrated in figure 6, an MBMS Aware Application that is registered when a requested file is successfully received is notified of the availability of the new file via the `fileAvailable()` API.

Figure 6.2.3.11.1-1 illustrates what happens when an application registers to consume File Delivery Application Services with a non-zero `registrationValidityDuration`, asks different files to be captured (possibly from different File Delivery Application Services), and then de-registers.

During the registration validity duration period following the application de-registration, the MBMS client collects files matching the outstanding capture requests from the application and keeps the files in its cache while the application is not currently registered.

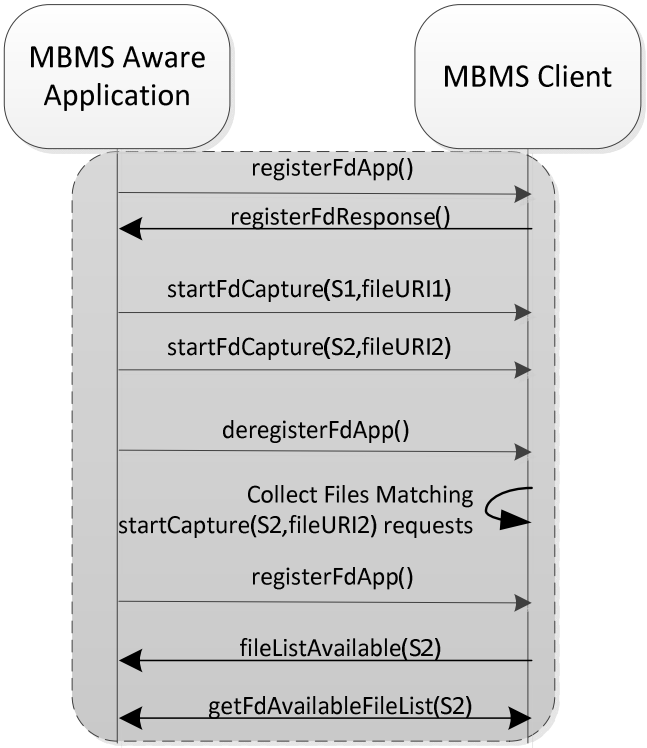

Figure 6.2.3.11.1-1: Sequence diagram for notifying the application about collected files

6.2.3.11.2 Parameters

The parameters for the `fileListAvailable()` API are:

- `string serviceId` – identifies the File Delivery Application service.

6.2.3.11.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` mode.

The MBMS client does have an `_fileURIStatus[]` entry for which the `_notified` flag is set to false

6.2.3.11.4 Expected MBMS Client Actions

The MBMS client invokes the `fileListAvailable()` notification with the following parameters:

- The `serviceId` is set to the `_serviceID`.

6.2.3.11.5 Usage of Method for Application

The application receiving this notification may then ask the client on the status of the file list using the `getFdDownloadStateList()` method defined in clause 6.2.3.16.

6.2.3.11.6 Post-Conditions

No state change is involved.

6.2.3.12 Getting the List of Available Files

6.2.3.12.1 Overview

This clause defines the `getFdAvailableFileList()` request.

As illustrated in Figure 6.2.3.12.1-1, once the MBMS Aware Application re-registers with the MBMS client, the MBMS client may indicate through the `fileListAvailable()` callback function to the application know that a list of files have been received for a service and are now ready to be accessed.

That application can then invoke the `getFdAvailableFileList()` API to retrieve information on these received files.

6.2.3.12.2 Parameters

The parameters for the `getFdAvailableFileList()` API are:

- Input parameters:
  - `string serviceId` – identifies the File Delivery Application Service on for which the application requests the available file list.
- Output parameters
  - A list for records, each containing:
    - `string fileUri` – identifies the file captured on the service identified in serviceId.
    - `string fileLocation` – identifies the location where the MBMS Aware Application can find the collected file.
    - `string contentType` – indicates the MIME type for the file identified in the fileUri.
    - `unsigned long availabilityDeadline` – signals a deadline in seconds when the file stored at the fileLocation will be removed from the MBMS client storage location, if applicable.

6.2.3.12.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state for the `serviceId`.

The MBMS client has indicated through the `fileListAvailable()` callback function to the application know that a list of files have been received for a service and are ready to be accessed.

The MBMS client has pre-configured an internal parameter `defaultAvailabilityDeadline` defining the time a file that is kept in the MBMS client owned storage location.

6.2.3.12.4 Usage of Method for Application

The application may use this method in order to retrieve information on the received files.

Once the application receives the response, the application may access the files that are announced by the parameters in the callback function. If the storage is not defined by the `locationPath` the application should access the file before the announced `availabilityDeadline`.

6.2.3.12.5 Expected MBMS Client Operation

When this method is invoked, the MBMS client provides the file parameters as defined above.

6.2.3.12.6 Post-Conditions

The MBMS client has notified the application about all accessible files.

6.2.3.13 Stop File Delivery Capture

6.2.3.13.1 Overview

This clause defines the `stopFdCapture()` request.

The application can make `startFdCapture()` calls to define fileURIs for the files to be received within an MBMS Service. The application may use the use the `stopFdCapture()`request to undo such a call and stop capturing such requested files. This is shown in Figure 6.2.3.13.1-1.

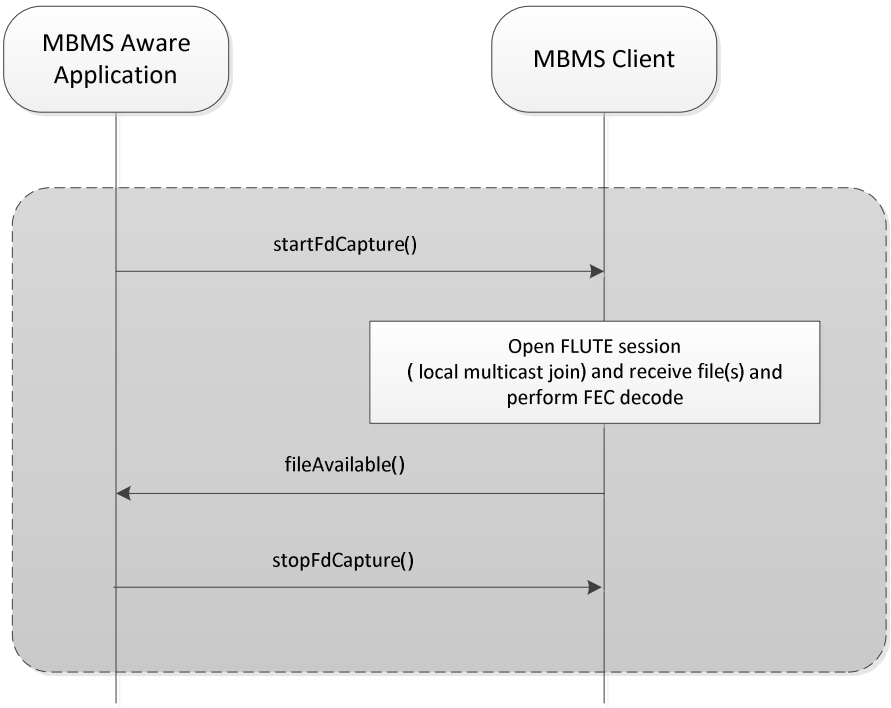

Figure 6.2.3.13.1-1: File Delivery Application Stop Capture

6.2.3.13.2 Parameters

The parameters for the `stopFdCapture()` API are identical to the parameters for `startFdCapture()` as defined in clause 6.2.3.7.2.

6.2.3.13.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state.

6.2.3.13.4 Usage of Method for Application

The application may use this request to undo such a call and stop capturing such requested files.

The application should cache the requested fileURIs and use the `stopFdCapture()` API to signal to the MBMS client when the application no longer wishes to receive files matching the fileURIs on earlier capture requests.

If the MBMS Aware application has not properly cached the list of fileURIs on its outstanding `startFdCapture()` requests, the application should invoke the `getFdActiveServices()` API described in clause 6.2.3.4 to re-synchronize on its outstanding `startFdCapture()` requests.

6.2.3.13.5 Expected MBMS Client Actions

Upon receiving a `stopFdCapture()` the MBMS client updates its internal parameters and sends notifications, if applicable.

6.2.3.13.6 Post-Conditions

Entries in the `_fileCaptureRequest[]` record may be removed. If by this action the `_fileCaptureRequest[]` gets empty, the MBMS client changes to `REGISTERED` state.

6.2.3.14 Getting the list of outstanding fileURIs being captured

6.2.3.14.1 Overview

This clause defines the `getFdActiveServices()` request.

Figure 6.2.3.14.1-1 illustrates that the application may also invoke the `getFdActiveServices()` API to retrieve the `fileURI` for outstanding `startFdCapture()` requests.

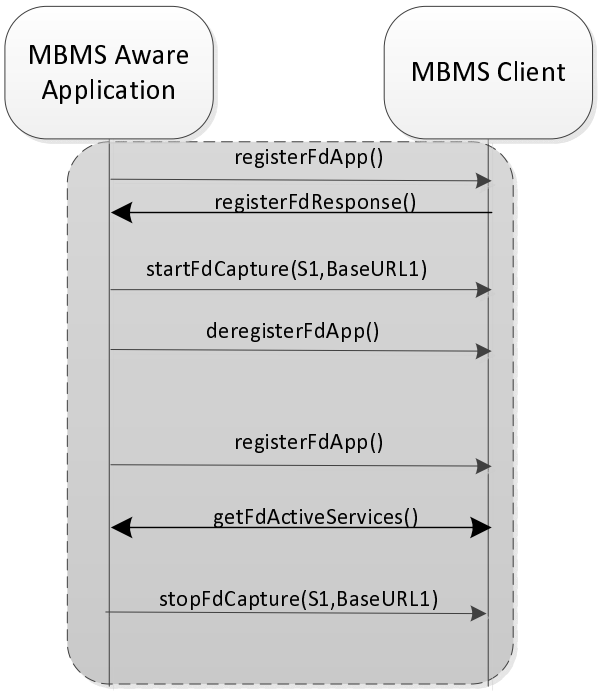

Figure 6.2.3.14.1-1: Sequence diagram for an application to collect info on outstanding `startFdCapture()` requests

6.2.3.14.2 Parameters

The parameters for the `getFdAvailableFileList()` API are:

- Input parameters:
  - `string serviceId` – identifies the File Delivery Application Service on for which the application requests outstanding `startFdCapture()` requests.
- Output parameters:
  - A list for records, each containing:
    - `string fileUri` – as defined in clause 6.2.3.7.2.

6.2.3.14.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state.

6.2.3.14.4 Usage of Method for Application

The application may also invoke the `getFdActiveServices()` API to retrieve the `fileURI` for outstanding `startFdCapture()` requests.

It is recommended that especially after a new registration or once a more recent `startFdCapture()` with a BaseURL superseded an earlier `startFdCapture()` with an `AbsoluteURL`.

6.2.3.14.5 MBMS Client Actions

When this method is invoked, the MBMS client provides the parameters in the call.

6.2.3.14.6 Post-Conditions

This call does not change any state or internal parameters of the MBMS client.

6.2.3.15 Notification on state change for files

6.2.3.15.1 Overview

This clause defines the `fileDownloadStateUpdate()` callback function.

As illustrated in Figure 6.2.3.15.1-1, after an MBMS Aware Application registers with the MBMS client and requests that files are to be captured, the MBMS client may issue `fileDownloadStateUpdate()` notifications to an application to signal that the state the MBMS client maintains for file(s) received or being received for the application has changed.

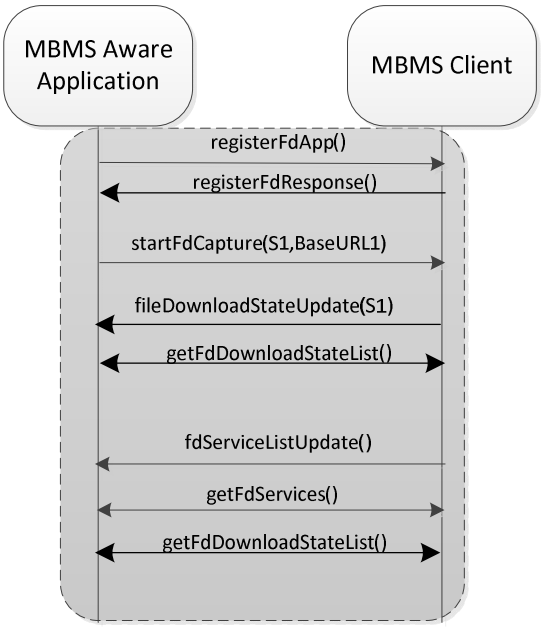

Figure 6.2.3.15.1-1: Sequence diagram for notifying the application about changes to the state of files being collected

6.2.3.15.2 Parameters

The parameters for the `fileDownloadStateUpdate()` callback function are:

- `string serviceId` – identifies the File Delivery Application Service on for which a new status is indicated.

6.2.3.15.3 Pre-Conditions

The MBMS client is in the `CAPTURE_NOTIFY` state.

6.2.3.15.4 Expected MBMS Client Actions

The MBMS client may invoke this method whenever the in the internal `_fileURIStatus[]` was changed.

6.2.3.15.5 Usage of Method for Application

If the application receives this callback notification, it should issue a `getFdDownloadStateList()`.

6.2.3.15.6 Post-Conditions

No state change is happening, but the application is aware that it can receive updated information from the MBMS client.

6.2.3.16 Getting the state on file(s) received or being received

6.2.3.16.1 Overview

This clause defines the `getFdDownloadStateList()`request function.

An MBMS Aware application may be interested to retrieve the current state for files downloaded or being downloaded by the MBMS client on behalf of that application. The application may choose to request this information in response to a notification from the MBMS client of such state change via a `fileDownloadStateUpdate()` notification, but also in other cases.

6.2.3.16.2 Parameters

The parameters for the `getFdDownloadStateList()` API are:

- Input parameters:
  - `string serviceId` – identifies the File Delivery Application Service on for which the application requests the available file list.
- Output parameters:
  - A list for records, each containing:
    - `string fileUri` – identifies the file captured on the service identified in `serviceId`.
    - `DownloadState state` – state of the file identified by the `fileUri`. The value can be as follows:
      - `FD _REQUESTED` - capture is requested, but file is neither scheduled nor is reception in process. This may for example be the case if the reception has failed, so the capture request stays open.
      - `FD_SCHEDULED` - scheduled, if a file schedule is delivered and a file is scheduled, but not yet received.
      - `FD_IN_PROGRESS` – the reception of the file is in progress.

6.2.3.16.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state.

6.2.3.16.4 Usage of Method for MAA

An MAA may be interested to retrieve the current state for files downloaded or being downloaded by the MBMS client on behalf of that MAA. The MAA may choose to request this information in response to a notification from the MBMS client of such state change via a `fileDownloadStateUpdate()` notification.

The MAA may also detect via updated service definition information (i.e., via a `fdServiceListUpdate()` followed by a `getFdServices()`) that a file previously advertised on an earlier `getFdServices()` and which the MAA requested to be capture is no longer described on the information retrieved via the latest `getFdServices()`, and the MAA did not receive a `fileAvailable()` or a `fileDownloadFailure()` reporting the successful or failed reception of the requested file, respectively. This could happen because the requested

file is no longer advertised as available for request (there is no current of future transmission for the file described on a `fileSchedule` in the schedule description fragment), but the file is still pending file repair.

An interested MAA can request information on the current state for files requested to be downloaded by the MBMS client on behalf of that MAA by invoking the `getFdDownloadStateList()` API.

6.2.3.16.5 Expected MBMS Client Actions

When this API call is received, the MBMS client provides the state of all files for which a capture request is issued.

6.2.3.16.6 Post-Conditions

There is no state change involved in the client.

6.2.3.17 Notification of updates to the service definition

6.2.3.17.1 Overview

This clause defines the `fdServiceListUpdate()`callback function.

As illustrated in Figure 6.2.3.15.1-1, after an MBMS Aware Application registers with the MBMS client and possibly requests that files are to be captured, the MBMS client may issue `fdServiceListUpdate()` notifications to an application to signal that there have been changes to the definition of File Delivery Application Services associated with the service classes the application has registered with the MBMS client.

6.2.3.17.2 Parameters

none

6.2.3.17.3 Pre-Conditions

The MBMS client is in `REGISTERED` or `CAPTURE_NOTIFY` mode.

6.2.3.17.4 Expected MBMS Client Operation

The MBMS client invokes this notification when there has been a change to the parameters reported to the application in response to a `getFdServices()` API.

6.2.3.17.5 Usage of Method for Application

In response to a `fdServiceListUpdate()` API notification from the MBMS client, the MBMS Aware application should invoke a `getFdServices()` API and process the updated information accordingly.

6.2.3.17.6 Post-Conditions

There is no change in the client state.

6.2.3.18 Notification of File Delivery Application Service errors

6.2.3.18.1 Overview

This clause defines the `fdServiceError()` callback function.

As illustrated in Figure 6.2.3.18.1-1, the `startFdCapture()` request from an MBMS Aware Application may not be served, so the MBMS client will send a failure indication via the `fdServiceError()` to signal the error code for the result of processing the application's `startFdCapture()`.

Figure 6.2.3.18.1-2 also illustrates that the `fdServiceError()` is used to signal the error code for the result of processing the application's `stopFdCapture()` request.

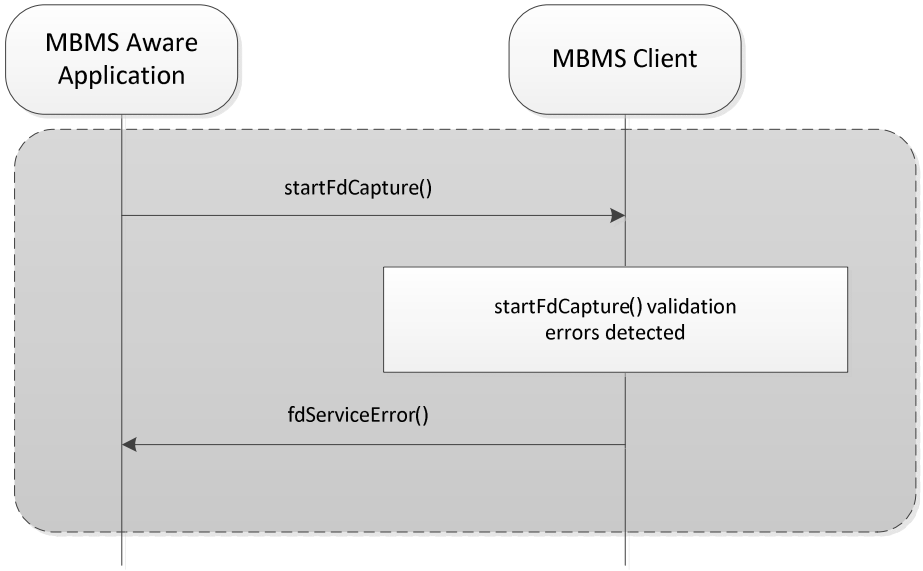

Figure 6.2.3.18.1-1: Signaling errors with the `startFdCapture()` request from the MBMS Aware Application

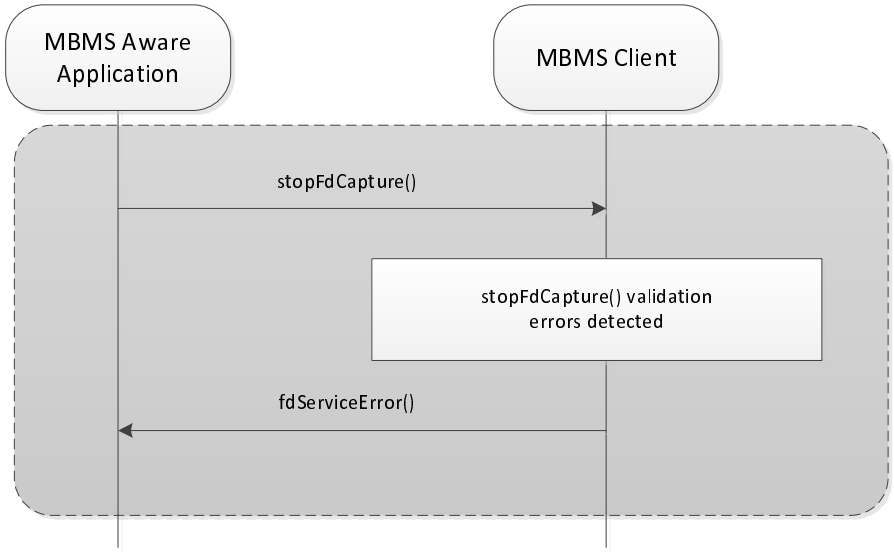

Figure 6.2.3.18.1-2: Signaling errors with the `stopFdCapture()` request from the MBMS Aware Application

6.2.3.18.2 Parameters

The parameters for the `fdServiceError()` callback are:

- `string serviceId` – identifies the File Delivery Application service for which this notification applies.
- `FdErrorCode errorCode` – identifies the error code for the reason causing the `startFdCapture()` or the `stopFdCapture()` request for the serviceId and fileUri to fail. The available error codes are:
  - `FD_INVALID_SERVICE` – signals that `serviceID` defined on the `startFdCapture()` or the `stopFdCapture()` request is not currently defined or it is not associated with the service classes with the MBMS Aware Application is registered.
  - `FD_DUPLICATE_FILE_URI` – signals that fileUri defined on the `startFdCapture()` request has already been requested on a previous `startFdCapture()` request. This is a duplicate request and the previous request is still in effect, i.e., impact to that earlier request. The MBMS client will not signal this error for the same condition on a `stopFdCapture()` request.

- `FD_AMBIGUOUS_FILE_URI` – signals that `fileUri` defined on the `startFdCapture()` or the `stopFdCapture()` request creates ambiguity with a previously issued `startFdCapture()` or `stopFdCapture()` request. See clauses 6.2.3.7 and 6.2.3.13 for details on the conditions when this error code is generated for the `startFdCapture()` and the `stopFdCapture()` request, respectively.
- `FD_STOP_FILE_URI_NOT_FOUND` – signals that the indicated `fileURI` does not match an outstanding `startFdCapture()` request.
- `FD_UNKNOWN_ERROR` – signals an error condition not explicitly identified.

- `string errorMsg` – may provide additional textual description of the error condition.

6.2.3.18.3 Pre-Conditions

The MBMS client is in `REGISTERED` or `CAPTURE_NOTIFY` mode.

6.2.3.18.4 Expected MBMS Client Actions

If a `startFdCapture()` or a `stopFdCapture()` request from an MBMS Aware Application may not be served, the MBMS client will send a failure indication via the `fdServiceError()` to signal the error code for the result of processing

6.2.3.18.5 Usage of Method for Application

If the application receives such a notification, it should check the capture requests and possibly also invoke the `getFdActiveServices()` API call to re-synchronize on its outstanding `startFdCapture()`requests.

6.2.3.18.6 Post-Conditions

No state change applies.

6.2.3.19 Notification on storage limitations

6.2.3.19.1 Overview

This clause defines the `insufficientStorage()` callback function.

As illustrated in Figure 6.2.3.19.1-1, once a file is to be received for an MBMS Aware Application (at a scheduled transmission time for the respective File Delivery Application Service), the FDT for the FLUTE session for that service will signal the size for that file.

When the MBMS client detects that not enough storage is available on the UE to receive the file, the MBMS client will send the warning indication via the `insufficientStorage()` API to signal the application of the low storage condition. The application may be able to clean up some of its own files or alert the user to clean up storage space on the UE.

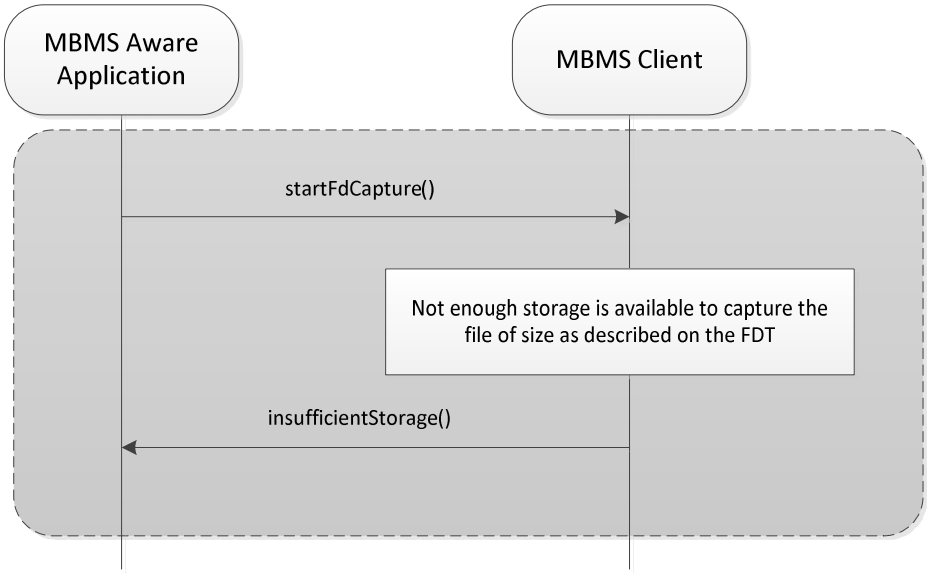

Figure 6.2.3.19.1-1: Signaling a low storage level condition impacting a file download

6.2.3.19.2 Parameters

The parameters for the `insufficientStorage()` callback are:

- `string serviceId` – identifies the File Delivery Application service for which this notification applies.
- `string fileUri` – identifies the file being transmitted on the File Delivery Application Service in serviceId which the MBMS client has started to capture and that will fail download because of insufficient storage on the UE.
- `StorageLocation locationPath` – identifies a local directory available on the device storage, which the application can access and where successfully collected files can be copied/moved before notifying that the file is available to the application. The storage location is a string pointing to a directory, or it may be empty, if no location is provided.
- `unsigned long storageNeeded` – indicates the additional storage space that needs to be cleared on the storage Path to enable the download of the file in fileURI to succeed.
- `string errorMsg` – may provide additional textual description of the error condition.

6.2.3.19.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state.

6.2.3.19.4 Expected MBMS Client Actions

The `insufficientStorage()` API is invoked for the applications that are currently registered at the time that the low storage condition is detected.

6.2.3.19.5 Usage of Method for Application

If the MBMS-aware application receives such a notification, it should take appropriate measures to enable to continue the service. This may include to change the service location, clear some storage autonomously or to communicate with the user to release storage.

6.2.3.19.6 Post-Conditions

No state changes occur.

6.2.3.20 Notification on storage access issues

6.2.3.20.1 Overview

This clause defines the `inaccessibleLocation()` callback function.

As illustrated in Figure 6.2.3.20.1-1, the locationPath where the MAA registered to have its requested files copied may not be available (e.g., SD card not inserted/locked). When the MBMS client detects that the register locationPath is not accessible, the MBMS client will send the warning indication via the `inaccessibleLocation()` API to signal the MAA of the storage access limitation; this can be done at different times, e.g., following a `startFdCapture()` as illustrated in Figure 6.2.3.20.1-1. The MAA may select an alternative `locationPath`, or prompt the user to choose another `locationPath`. The MAA can notify the MBMS client of the new `locationPath` via the `setFdStorageLocation()` API.

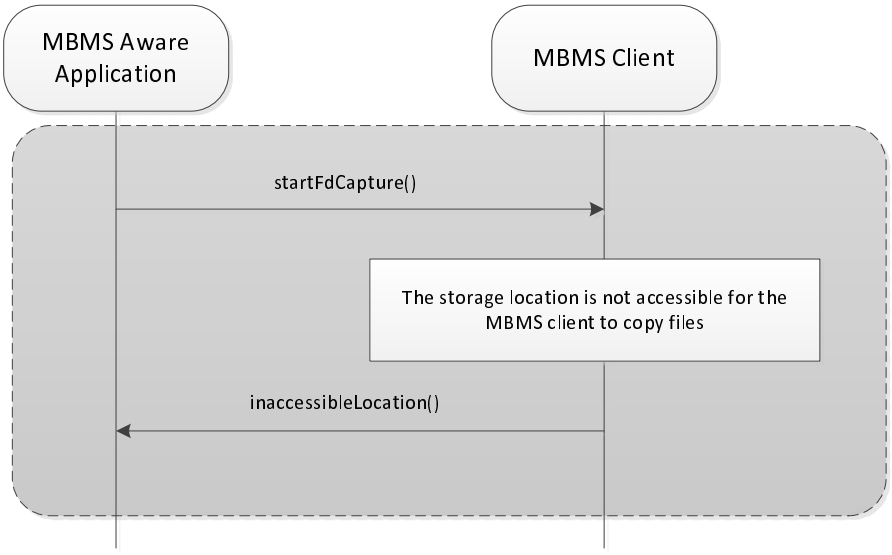

Figure 6.2.3.20.1-1: Signaling a storage access condition limitation impacting file download

6.2.3.20.2 Parameters

The parameters for the `inaccessibleLocation()`callback are:

- `string serviceId` – identifies the File Delivery Application service for which this notification applies.
- `StorageLocation locationPath` – indicates the storage location where files are to be stored per the registered locationPath at the time that the download started, which is not accessible by the MBMS client.
- `string errorMsg` – may provide additional textual description of the error condition.

6.2.3.20.3 Pre-Conditions

The MBMS client is in `CAPTURE_NOTIFY` state.

6.2.3.20.4 Expected MBMS Client Actions

The `inaccessibleLocation()` API is only invoked for the applications that are currently registered at the time that the storage inaccessible condition is detected.

6.2.3.20.5 Usage of Method for Application

If the application receives this notification, the application should take appropriated measures. It may for example change the location or may alert the user that the storage is not accessible.

6.2.3.20.6 Post-Conditions

No state change is applied, but the MBMS client may fail to receive files .

6.2.3.21 Checking the version for File Delivery Application Service interface

6.2.3.21.1 Overview

This clause defines the `getVersion()` request function.

6.2.3.21.2 Parameters

The parameters for the `getVersion()` API call are:

- `string version` – identifies the version of the MBMS clients API implementation.

6.2.3.21.3 Pre-Conditions

The MBMS client may be in any state.

6.2.3.21.4 Operation of Method in MBMS Client

The `getVersion()` API returns the version of the implemented APIs of the MBMS client.

6.2.3.21.5 Usage of Method for Application

In order for the MBMS Aware Application to know the version of the File Delivery Application Service interface, the getVersion() API may be used. If the version number is not supported by the application, it should deregister and not use the API.

6.2.3.21.6 Post-Conditions

Note state change applies.

6.3 DASH Streaming Service API

6.3.1 Introduction

The DASH Streaming Service API provides MBMS Aware Applications with interfaces to manage the reception of DASH Media Presentations delivered over DASH-over-MBMS services that are built on the Download Delivery Method. This API is intended to support live DASH streaming applications.

The IDL for the DASH Streaming Service API is defined in Annex B.3.

6.3.2 MBMS Client State Model for DASH Streaming

6.3.2.1 Overview

Figure 6.3.2-1 provides an informative client state model in order to appropriately describe the messages on the DASH streaming service API. Four different states are defined. State changes may happen based on:

- Calls from the MAA or the DASH client
- Information provided by the MBMS User Service (USD, schedule, FDT, file complete)
- Changes in the reception conditions

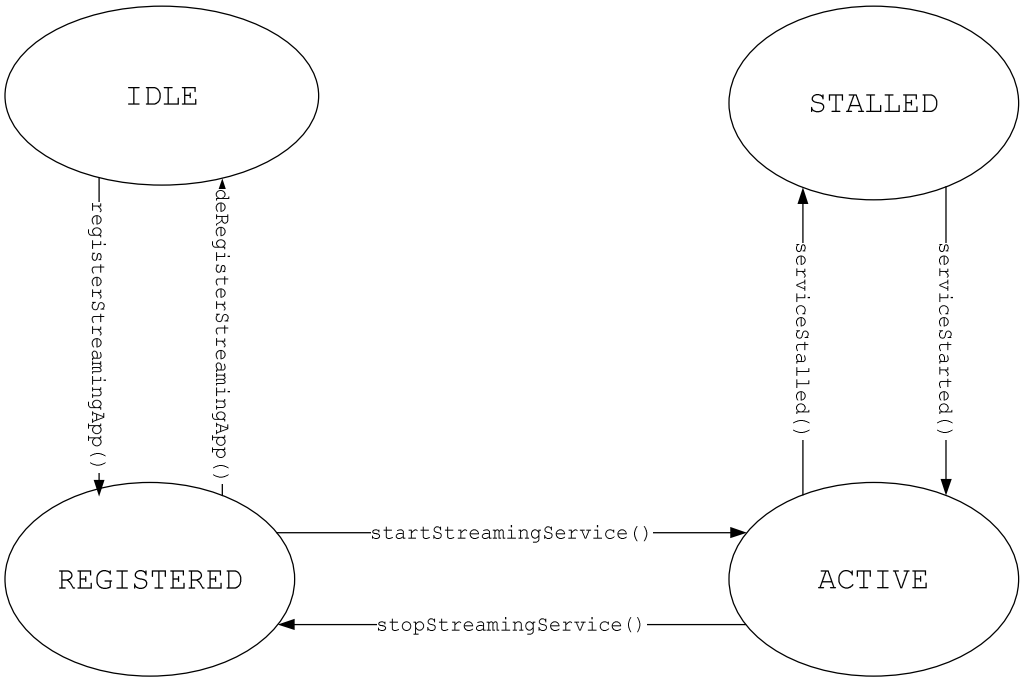

Figure 6.3.2-1: State Diagram

Table 6.3.2-1 defines states for the MBMS client. Detailed descriptions are provided in the following subclauses .

Table 6.3.2-1: States of MBMS Client

States and Parameters	Definition
IDLE	In this state the MBMS client does not have a registered MAA and it may not keep the service definition up to date. For more details see clause 6.3.2.3.
NON_AVAILABLE	In this state the MBMS client is not available and an MAA cannot register with the MBMS client.
REGISTERED	In this state the MBMS client has registered the MAA, it may keep the service definition up to date, and it may be providing file capture services to the MAA(s). In this state the MBMS client sends callback notifications to the MAA. For more details see clause 6.3.2.4.
ACTIVE	In this state the MBMS client provides all services to of the REGISTERED state and also provides the streaming service to the MAA. For more details see clause 6.3.2.5.
STALLED	In this state the MBMS client provides all services to of the REGISTERED state, but the streaming services is at least temporarily stalled. For more details see clause 6.3.2.6.

6.3.2.2 MBMS Client Internal parameters

The MBMS client maintains internal parameters as defined in Table 6.3.2.2-1. Note that the parameters are conceptual and internal and only serve for the purpose to describe message generation on the API calls.

Table 6.3.2.2-1: Parameters of MBMS Client for DASH Streaming Service

Internal Parameters		Definition
_app[]		The MBMS client maintains a parameter set per registered app
	_appId	A unique ID provided by the application and assigned to the app.
	_serviceClass[]	A list of service classes identifying the services the application has access to.

Internal Parameters	Definition
_registrationValidityDuration	A period of time following the application de-registration over which the MBMS client continues to capture files for the application, see clause.
_service[]	The MBMS client maintains a parameter list per service. In this context the list is assigned also to one app, but an implementation may share the internal parameter list assigned to a service across multiple apps.
_serviceID	The service ID for a Streaming Application service over which the MBMS client collects files for the application.
_serviceClass	The service class associated with the Streaming Application service assigned the Service ID.
_serviceLanguage	The language of the service
_serviceName[] _name _lang	The service name, possibly expressed in different languages.
_serviceBroadcastAvailability	The service broadcast availability for the client. Three different types are defined: BROADCAST_AVAILABLE – UE is in broadcast coverage BROADCAST_UNAVAILABLE – UE is outside of broadcast coverage SERVICE_UNAVAILABLE – The service is unavailable for the UE
_MPD _IS[] _mpdURI	The latest MPD associated to the service The Initialization Segments for the Media Presentation The URI which is provided to the application for initiating the DASH Media Presentation.
_sessionSchedule[] _start _stop	Documents the session schedule for this session. Only sessionSchedule records should be included for which the value of the _stop time is in the future.

6.3.2.3 MBMS Client Operation in IDLE state

In the IDLE state, the MBMS client may listen to the User Service Bundle Description and may collect information. However, no binding with the MAA is in place.

When the registerStreamingApp() as defined in subclause 6.3.3.2 is invoked, then:

1) The MBMS client checks the input parameters for consistency and sets the internal variables:

   a) If the functions of the MBMS client is not accessible, the MBMS client throws a FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE result code in the registerStreamingResponse() as defined in subclause 6.3.3.3 and abort the following steps and may at least temporarily move in NOT_AVAILABLE state.

   b) If appId is an empty string then the MBMS client throws a MISSING_PARAMETER result code in the registerStreamingResponse() as defined in subclause 6.3.3.3 and abort the following steps and stays in IDLE mode. If not, the MBMS client sets the internal variable _appId to the value of the parameter.

   c) The MBMS client adds each entry in the serviceClassList parameter to its _serviceClass[] record. Note that the serviceClassList parameter may contain an empty service class entry. If an empty service class is provided the MBMS client considers the MAA to be registered with a service class that is also empty and only allow the MAA to have access to DASH Streaming Application Services that are not associated with a serviceClass (i.e., the USD for these services do not have a serviceClass defined).

d) On receiving a `registerStreamingApp()` following a `deregisterStreamingApp()`, the MBMS client updates the `serviceClassList` to its `_serviceClass[]` record in the same way described for the `setStreamingServiceClassFilter()` method.

e) If `callBack` is defined, the MBMS client uses the interfaces in the callback parameter of the `registerStreamingApp()` interface to send notification of event occurrences to the MAA.

2. generates a response `registerStreamingResponse()` as defined in subclause 6.3.3.3 and changes to `REGISTERED` state as defined in clause 6.3.2.4:

a) If the MBMS client functions cannot be activated for any reason, especially if the Streaming Delivery Application Service API did not find an MBMS client available on the UE on which the MAA is running, the `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE` registration response code is sent. The MBMS client may provide a `message`.

b) If the MAA did not provide a mandatory parameter the MBMS client functions cannot be activated, the `MISSING_PARAMETER` registration response code is sent.

c) If the MBMS client functions can be activated, then:

i) the `RegResponseCode` is set to `REGISTER_SUCCESS` registration response code;

ii) a `message` may be generated.

d) Sends the response with the above parameters.

3. If the MBMS client functions can be activated and the response is sent with a `REGISTER_SUCCESS`, then MBMS client is in `REGISTERED` state and uses the `REGISTERED` parameters to provide the list of matching streaming delivery services using the information in the User Service Description (USD). If the response is sent with a `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE`, then MBMS client is in `NOT_AVAILABLE` state. If the response is sent with a `MISSING_PARAMETER`, then MBMS client is in `IDLE` state.

If the MBMS client receives the `getVersion()` API call as defined in clause 6.3.3.13, it shall return version 1.0.

6.3.2.4 MBMS Client Operation in REGISTERED state

For each registered MAA and the assigned parameters according to Table 6.3.2.2-1, the MBMS client uses the information in the User Service Description as well as its internal state information for the MAA in `_app[]` in the service class list `_serviceClass[]` to collect and keep up-to-date all internal information for the services of interest for the app, i.e. those that are member of any service class for which the MAA has interest.

For each MBMS user service for which the USD as defined in TS 26.346 [5] is available in the MBMS client for the service classes registered by the MAA in `_serviceClass[]` and which is identified as a DASH-over-MBMS service according to the definition in TS 26.346 [5], clause 5.6, one service record in the internal parameter `_service[]` is defined in the MBMS client and continuously updated whenever a new USD is available:

- For each **`userServiceDescription.name`** element, a (`name`, `lang`) pair is generated and added to the `_serviceName[]` list with `_name` set to the value of the USD element, and if present, the `_lang` set to the value of the associated `@lang` attribute. If no `@lang` attribute is present, the `_lang` parameter is set to an empty string.

- If the attribute **`userServiceDescription`**`@serviceClass` is present, the value of this attribute is assigned to `_serviceClass`. If not present, the `_serviceClass` is set to an empty string.

- The value of the attribute **`userServiceDescription`**`@serviceId` is assigned to `_serviceId`.

- If the attribute **`userServiceDescription`**`@serviceLanguage` is present, the value of this attribute is assigned to `_serviceLanguage`. If not present, the `_serviceLanguage` is set to an empty string.

- The MPD metadata fragment referenced by either the **`r9:mediaPresentationDescription`** element or by an a **`r12:appService`** referencing an MPD and conforming to TS 26.247 [7] is extracted by the MBMS client. The contained MPD is stored in the `_MPD` parameter and the Initialization Segments are stored in the `_IS[]`. The `_mpdURI` parameter is generated at which location the MPD will be made available.

- The _serviceBroadcastAvailability is continuously updated set it to
  - BROADCAST_AVAILABLE, if broadcast is available (if the UE is in broadcast coverage of the service),
  - BROADCAST_UNAVAILABLE, if broadcast is not available (if the UE is NOT in broadcast coverage of the service).
- If the **userServiceDescription.schedule** element is present then the MBMS client uses the information in the schedule description fragment to generate the internal _sessionSchedule[] list and keep up to date as a result of USD updates. The MBMS client shall only include _sessionSchedule[] records if the _stop value is in the future.

If updates are provided and added to the _service[] parameter, the MBMS client should send a streamingServiceListUpdate() callback as defined in clause 6.3.3.6.

When the getStreamingServices() method is received as defined in clause 6.3.3.4, the MBMS client sets the parameters as follows:

- If the _service[] list is empty, the list is empty.
- For each MBMS user service in the service[] list, one service record is generated as follows:
  - The value of the attribute _serviceId is assigned to serviceId.
  - The value of the attribute _serviceClass is assigned to serviceClass.
  - The value of _serviceLanguage is assigned to serviceLanguage.
  - For each record in the _serviceName[] one serviceNameList entry is generated and:
    - the name is set to the value _name,
    - the name is set to the value _name,
  - The value of _serviceBroadcastAvailability is assigned to serviceBroadcastAvailability.
  - The _mpdURI is assigned to mpdURI.
  - If at least one _sessionSchedule[] record is present then:
    - The activeDownloadPeriodStartTime is set to the value of earliest _start time of any entry in the _sessionSchedule[].
    - The activeDownloadPeriodStopTime is set to the value of the _stop time of the entry selected earliest start time.
  - If no _sessionSchedule[] record is present:
    - The activeDownloadPeriodStartTime is set to 0.
    - The activeDownloadPeriodStopTime is set to 0.

When the setStreamingServiceClassFilter() as defined in clause 6.3.3.5 is received, the MBMS client runs the following steps:

- It replaces the internal variable _serviceClass[] with the parameter values provided in serviceClassList.
- The MBMS client dis-associates the service classes previously associated with the MAA that are not included on this list.
- The MBMS client associates the service classes not previously associated with the MAA that are newly included on this list.

- The MBMS client issues a `streamingServiceListUpdate()` notification as defined in clause 6.3.3.6 to the MAA to notify of this effect.

When the `startStreamingService()` method as defined in clause 6.3.3.7 is received with a parameter `serviceID`, the MBMS client runs the following steps:

- The MBMS client checks for errors and if necessary, the `streamingServiceError()` notification as defined in clause 6.3.3.12 is initiated. Specifically, if the MBMS client does not find a matching `serviceId` in its internal `_service[]` record, it responds with error code `STREAMING_INVALID_SERVICE`. Otherwise it may use the error code `STREAMING_UNKNOWN_ERROR`. An error message may be provided in the `errorMsg` string.
- If the service with the `serviceId` parameter can be started:
  - The MBMS client uses the MPD in the `_MPD` parameter and the remaining associated metadata to offer a valid media presentation to the DASH client by providing a DASH server in the MBMS client. For different options to provide such a service, refer to clause 7.
  - The URL to the MPD that is exposed to the MAA for DASH consumption is stored in the internal variable `_mpdURI`. The MPD stored at this URI may be continuously updated, based on dynamic information received in the service announcement or inband MPD updates.
  - The MBMS client sends a `serviceStarted()` notification as defined in clause 6.3.3.8 with the `serviceId` being passed along with the notification.
  - The MBMS client moves to ACTIVE state as defined in clause 6.3.2.5.

Whenever there has been a change to the parameters reported to the MAA in response to a `getStreamingServices()` API, i.e. in the internal service class list `_serviceClass[]` to add a new service record to the list or a change in one of the following internal parameters in the service record in the `_serviceLanguage`, `_serviceName[]_serviceBroadcastAvailability`, or updates to the `_mpdURI`, the MBMS client notifies the MAA with `streamingServiceListUpdate()` as defined in clause 6.3.3.6.

When the `deregisterStreamingApp()` is received, all internal parameters for the MAA are cleared and the client moves to `IDLE` state.

6.3.2.5 MBMS Client Operation in ACTIVE state

In the `ACTIVE` state, the MBMS client carries out all actions as in the `REGISTERED` state.

The MBMS client continuously downloads the DASH resources and makes them available as announced in the MPD. For different options to provide such a service to the MAA and DASH client, refer to clause 7. The URL to the MPD that is exposed to the MAA for DASH consumption is stored in the internal variable `_mpdURI`. The MPD stored at this URI may be continuously updated, based on dynamic information received in the service announcement or inband MPD updates.

When the MBMS client receives a `stopStreamingService()` request as defined in clause 6.3.3.9 that matches an active service.

- The MBMS client checks for errors and if necessary, the `streamingServiceError()` notification as defined in clause 6.3.3.12 is initiated. Specifically, if the MBMS client does not find a matching `serviceId` in its internal `_service[]` record, it responds with error code `STREAMING_INVALID_SERVICE`. Otherwise it may use the error code `STREAMING_UNKNOWN_ERROR`. An errorMsg may be provided in the `errorMsg` string.
- the MBMS client stops providing the DASH resources at its DASH server, i.e. at the location announced in the MPD referenced by the `_mpdURI`.
- The MBMS client moves to `REGISTERED` state as defined in clause 6.3.2.4.

When the MBMS client receives a `stopStreamingService()` request as defined in clause 6.3.3.9 that matches an active service, the MBMS client terminates the download of the resources of this download delivery session and no longer makes it available at the indicated resources in the MPD. The MBMS client transitions to `REGISTERED` state

When the MBMS the internal parameter `_serviceBroadcastAvailability` transitions to `BROADCAST_UNAVAILABLE`, and no alternative delivery method is defined, or if the service is no longer available for other reasons (e.g. frequency conflict), then the service is stalled. In this case the MBMS client:

- No longer makes available the resources in the announced locations by the `_mpdURI` and the references therein.
- Sends a `serviceStalled()` notification as defined in clause 6.3.3.11, along with one of the following reasons:
  - `RADIO_CONFLICT` – indicates a frequency conflict, namely the service requested to be started via a `startStreamingService()` cannot be started at this time since the MBMS client is actively receiving another service on a different frequency band.
  - `END_OF_SESSION` – indicates that playback has reached the end of the scheduled transmission for the service as described by the schedule description fragment for the service. This should indicate that the advertised activeServicePeriodEndTime time has been reached.
  - `OUT_OF_COVERAGE` – indicates a UE mobility event to an area where the service with streamingSubtype set to `STREAMING_BC_ONLY` is not available via broadcast.
  - `STALLED_UNKNOWN_REASON` – indicates that another unspecified condition caused the service interruption.
- Transitions to the `STALLED` state as defined in clause 6.3.2.6.

6.3.2.6 MBMS Client Operation in STALLED state

In the `STALLED` state, the MBMS client carries out all actions as in the `REGISTERED` state.

In this state the MBMS client continuously monitors if the service can be made available again.

Once the service gets available again, the MBMS client:

- The MBMS client downloads the DASH resources and makes them available as announced in the MPD. For different options to provide such a service, refer to clause 7. The URL to the MPD that is exposed to the MAA for DASH consumption is stored in the internal variable `_mpdURI`. The MPD stored at this URI may be continuously updated, based on dynamic information received in the service announcement or inband MPD updates
- The MBMS client sends a `serviceStarted()` notification as defined in clause 6.3.3.8 with the `serviceId` being passed along with the notification.
- The MBMS client moves to `ACTIVE` state as defined in clause 6.3.2.5.

6.3.3 Methods

6.3.3.1 Overview

Table 6.3.3.1-1 provides an overview over the methods defined for the Streaming Delivery Application Service API. Different types are differentiated, namely state changes triggered by the MAA, status query of the MAA to the client, parameter updates as well as notifications from the client. The direction of the main communication flow between MAA (A) and MBMS Client (C) is provided.

Table 6.3.3.1-1: Methods defined for Streaming Delivery Application Service API

Method	Type	Direction	Brief Description	Section
`registerStreamingApp`	State change	A -> C	MAA registers a callback listener with the MBMS client	6.3.3.2
`deregisterStreamingApp`	State change	A -> C	MAA deregisters with the MBMS client	6.3.3.10
`startStreamingService`	State change	A -> C	Starts streaming service	6.3.3.7

Method	Type	Direction	Brief Description	Section
stopStreamingService	State change	A -> C	Stop streaming service	6.3.3.9
getStreamingServices	Status query	C <-> A	Get list of currently active services	6.3.3.4
getVersion	Status query	C <-> A	Retrieves the list of files previously captured for the MAA	6.3.3.13
setStreamingServiceClassFilter	Update to parameter list	A -> C	MAA sets a filter on file delivery services in which it is interested	6.3.3.5
registerStreamingResponse	Update to parameter list	C-> A	The response to the MAA streaming service register API	6.3.3.3
serviceStarted	Notification	C -> A	Notification to MAA when the streaming service started.	6.3.3.8
streamingServiceListUpdate	Notification	C -> A	Notification to MAA on an update of the available for DASH streaming delivery services	6.3.3.6
streamingServiceError	Notification	C -> A	Notification to MAA when there is an error with the download of service	6.3.3.12
serviceStalled	Notification	C -> A	Notification to MAA that download DASH segments failed	6.3.3.11

6.3.3.2 Registration

6.3.3.2.1 Overview

This clause defines `registerStreamingApp()` interface.

An MAA calls the `registerStreamingApp()` interface to register with the MBMS Client to consume streaming services. The `registerStreamingApp()` interface has two purposes:

1) It signals to the MBMS Client that an MAA is interested to consume MBMS DASH Streaming Services.

2) It allows the MAA to identify its callback listeners defined in the Streaming Service API for the MBMS Client to provide asynchronous notifications to the MAA on relevant events associated with streaming.

NOTE: Since some application development frameworks do not support callback functions, an MAA for these frameworks will not provide callback listeners in the `registerStreamingApp()` interface. Instead, the MAA will implement the necessary approach available on these frameworks to receive event notifications from the MBMS Client in place of callback functions. The notifications implemented on these frameworks will include the same information content as defined on the structures for the IDL callback functions.

Figure 6.3.3.2-1 shows the registration process

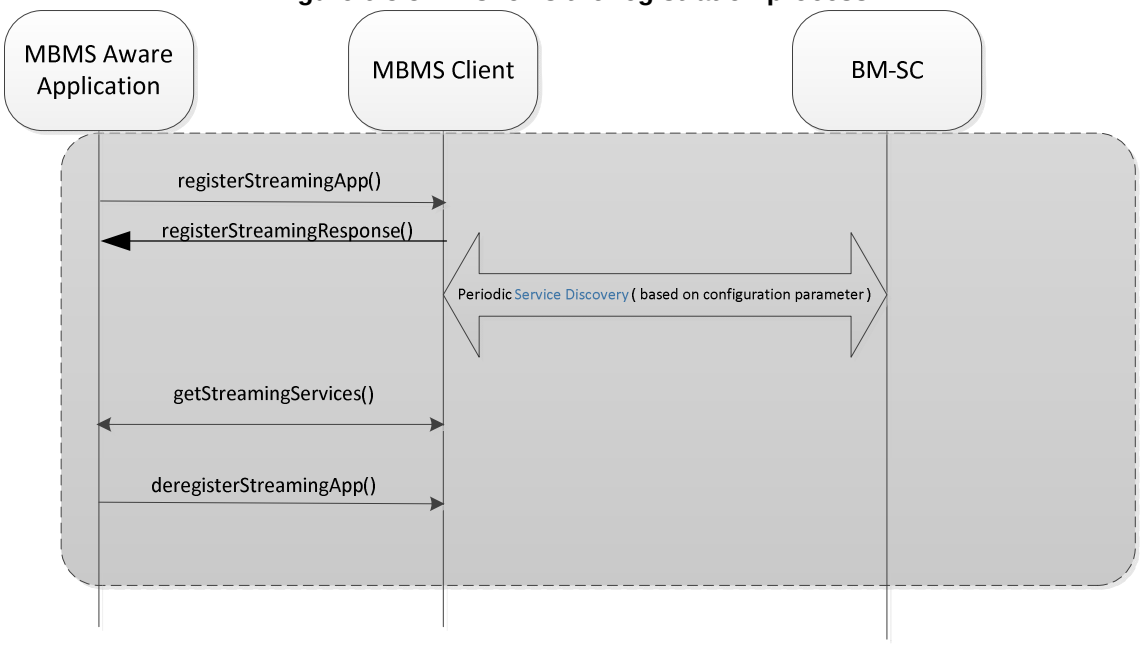

Figure 6.3.3.2-1: Application Registration sequence diagram

6.3.3.2.2 Parameters

The parameters for the `registerFdApp()` API are:

- **`string`** `appId` – provides a unique ID for the MAA registering with the MBMS client, which uses this identity to maintain state information for a particular MAA. The uniqueness of the ID is in the context of any MAA that may possibly register with MBMS client. Uniqueness is typically provided on platform level.
- **`any`** `platformSpecificAppContext` – a platform-specific context for the registering MAA that enables the MBMS client to get extra information about the MAA that may be need to enable the MAA to have access to MBMS services, e.g., to enable MAA authentication or to enable the MAA to communicate with the MBMS client via platform (e.g., HLOS) services.
- **`sequence<string>`** `serviceClassList` – provides a comma-separated list of service classes which the MAA is interested to register. Each service class string can be any string or it may be empty.
- **`ILTEFileDeliveryServiceCallback`** `callBack` – provides the MBMS client with the call back functions associated with DASH Streaming Application Service APIs for the registering MAA.

NOTE: The callback element in the IDL description is optional and only included when the MAA development framework supports programmatic callback interfaces. If callbacks are not supported on a given MAA development framework, the same information content as defined on the callback structures is to be provided to the MAA via the notification method available with that development framework when the respective condition is met.

6.3.3.2.3 Pre-Conditions

The MAA has assigned a unique application ID `appId` in the context of its operation (e.g., a smartphone HLOS) with the MBMS client.

The MAA is pre-configured with the set of service classes that allows it to consume the DASH Streaming Services associated with these service classes.

The MAA has access to a DASH Streaming client.

The MAA may use this method at launch or after a `deregisterStreamingApp()` has been called.

The MBMS client is in `IDLE` state.

6.3.3.2.4 Usage of Method for MAA

The MAA uses the method `registerStreamingApp()` to register with the MBMS Client to consume Streaming Services.

The MAA provides its `appId` and, if applicable, some platform specific MAA context, `platformSpecificAppContext`.

The MAA provides the set of service classes which the MAA is interested to register.

6.3.3.2.5 Expected MBMS Client Actions

When this method is invoked, the MBMS client registers the app, if possible. For more details refer to clause 6.3.2.3.

6.3.3.2.6 Post-Conditions

The MAA expects the `registerStreamingResponse()` as defined in clause 6.3.3.3.

6.3.3.3 DASH Streaming Application Service Registration Response

6.3.3.3.1 Overview

This clause defines `registerStreamingResponse()`call.

As illustrated in Figure 6.3.3.2-1, the MBMS client responds to an MAA call to the `registerStreamingApp()` API with a `registerStreamingResponse()` call back providing the result of the registration request.

6.3.3.3.2 Parameters

The parameters for the `registerStreamingResponse()` API are:

- `EmbmsCommonTypes::RegResponseCode` value – provides a result code on the registration request. The allowed values are:
  - `REGISTER_SUCCESS` – indicates that the registration has been processed successfully and the MAA can proceed with other API interactions with the MBMS client for Streaming Delivery Application Services.
  - `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE` – Indicates that the registration has failed since the Streaming Delivery Application Service API did not find an MBMS client available on the UE on which the MAA is running and no MBMS service will be available to the MAA.
  - `MISSING_PARAMETER` – indicates that the registration has failed since one or more of the required parameter was missing.
- `string message` – provides an associated text description of the error message. The message may be empty.

6.3.3.3.3 Pre-Conditions

The MBMS client has received a call via the `registerStreamingApp()` API as defined in clause 6.3.3.2.

6.3.3.3.4 Expected MBMS Client Actions

The MBMS client responds accordingly and depending on the response moves to one of the states: `IDLE`, `NOT_AVAILABLE`, or `REGISTERED`. For more details refer to clause 6.3.2.4.

6.3.3.3.5 Usage of Method for MAA

Once the MAA receives a the `registerStreamingResponse()` with the `RegResponseCode` set to `REGISTER_SUCCESS`, the MAA can proceed with other API interactions with the MBMS client.

If the MBMS client is temporarily in `NOT_AVAILABLE`, if the `registerFdResponse()` signaled a failure with a `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE`, the MAA may periodically recheck if the state of the MBMS client changes by retrying the `registerFdRequest()`API.

If the MBMS client is responding with `MISSING_PARAMETERS`, the MAA should fix the parameters and retry the `registerFdRequest()` API.

6.3.3.3.6 Post-Conditions

If the MBMS client functions cannot be activated and once the response is sent, then MBMS client is at least temporarily in `NOT_AVAILABLE` state.

If the MBMS client functions can be activated and respective response is sent, then MBMS client is in `REGISTERED` state with the `REGISTERED` parameters as set above.

6.3.3.4 Getting information on available DASH Streaming Application Services

6.3.3.4.1 Overview

This clause defines `getStreamingServices()` API call.

The `registerStreamingApp()` interface returns the complete list of available Streaming Services information. As illustrated in Figure 6.3.3.2-1, after a successful registration with the MBMS client, the MAA can use the `getStreamingServices()` API to discover the available Streaming Services associated with the service classes registered via the `registerStreamingApp()`.

6.3.3.4.2 Parameters

The `getStreamingServices()` API returns a list describing the available DASH Streaming Services, where each service is described by the following output only parameters:

- `sequence<ServiceNameLang> serviceNameList` – optionally provides a list of the service title name in possibly different languages. Each (name, lang) pair defines a title for the service on the language indicated.
  - `string name` – offers a title for the user service on the language identified in the lang parameter.
  - `string lang` – identifies a natural language identifier per RFC 3066 [10].
- `string serviceClass` – identifies the service class which is associated with the service.
- `string serviceId` – provides the unique service ID for the service. The uniqueness is among all services provided by the BMSC.
- `string serviceLanguage` – indicates the available language for the service and represented as an identifier per RFC 3066 [10].
- `EmbmsCommonTypes::ServiceAvailabilityType serviceBroadcastAvailability` – signals whether the UE is currently in the broadcast coverage area for the service.
  - The possible values are:
    - `BROADCAST_AVAILABLE` – if content for the service is broadcast at the current device location.
    - `BROADCAST_UNAVAILABLE` – if content for the service is not broadcast at the current device location.
    - SERVICE_UNAVAILABLE – if content for the service is at all available at the current device location.
- `string mpdUri` – provides an HTTP URL where the MPD for the DASH Streaming Application Service is hosted and available for DASH clients access.
- `EmbmsCommonTypes::Date activeServicePeriodStartTime` – signals the current/next active DASH Streaming Application Service start time, when DASH media segments and other resources start being broadcast over the air.
- `EmbmsCommonTypes::Date activeServicePeriodEndTime` – signals the current/next active DASH Streaming Application Service stop time, when DASH media segments and other resources stop being broadcast over the air.

6.3.3.4.3 Pre-Conditions

The MBMS client is in `REGISTERED` state.

6.3.3.4.4 Expected MBMS Client Actions

When this method is invoked, the MBMS client returns the streaming service parameters. For more details refer to clause 6.3.2.4.

6.3.3.4.5 U Usage of Method for MAA

The MAA should use this call right after the `registerStreamingResponse()` notification as defined in clause 6.3.3.3 is received or after the `streamingServiceListUpdate()` notification as defined in clause 6.3.3.6 is received.

The MAA should use the `serviceId` to identify the service in subsequent communication with the MBMS client to manage the streaming service.

The usage of the parameters `serviceNameList`, `serviceClass`, `serviceBroadcastAvailability`, and `serviceLanguage` is typically up to the MAA.

The `mpdURI` should be used by the MAA to initiate playback by initiating a DASH client. The MAA should assume that Media Presentation can be consumed by the DASH client without any further interaction with the MAA.

The parameters `activeServicePeriodStartTime` and `activeBroadacstPeriodEndTime` provides the MAA the ability to determine the current broadcast state for the service as follows:

- If the current time is such that `activeServicePeriodStartTime` ≤ `current time` ≤ `activeServicePeriodEndTime`, DASH content is being broadcast for the service at the current time.
- If the `activeServicePeriodStartTime` is in the future, there is currently no broadcast being made for the service, but broadcast transmission is currently scheduled to start at this advertised time.
- If the `activeServicePeriodStartTime` is set to zero, there is no currently defined broadcast schedule time for the service.

6.3.3.4.6 Post-Conditions

This call does not change the MBMS client state.

The MAA uses the `serviceId` to identify the service in subsequent communication with the MBMS client.

6.3.3.5 Updating the registered service classes

6.3.3.5.1 Overview

This clause defines `setStreamingServiceClassFilter()` call.

While an MAA is actively registered with the MBMS client to consume DASH Streaming Services, the MAA can call the `setStreamingServiceClassFilter()` API to update the list of service classes the MAA wants to be registered with, see Figure 6.3.3.5.1-1.

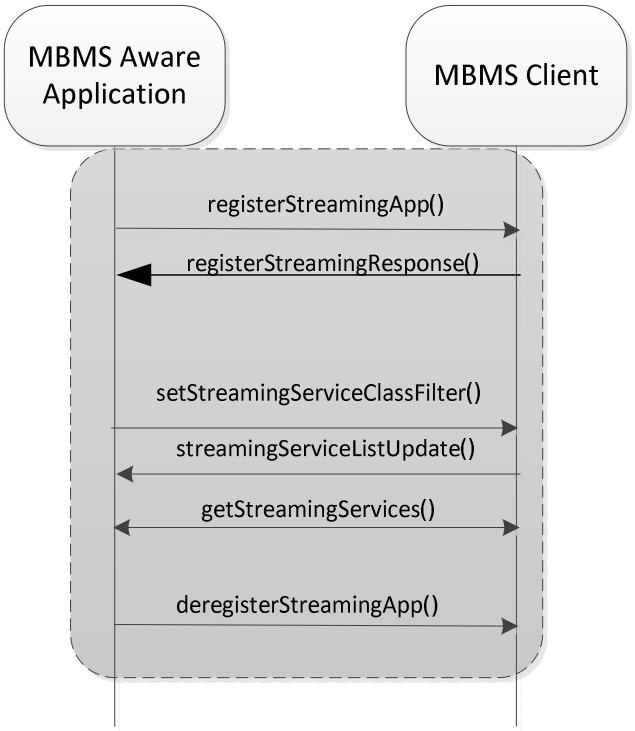

Figure 6.3.3.5.1-1: Sequence diagram for updating the registered service classes for an MAA

6.3.3.5.2 Parameters

The parameters for the `setStreamingServiceClassFilter()` method are:

- sequence<string> serviceClassList – see clause 6.3.3.2.2

6.3.3.5.3 Pre-Conditions

The MAA is actively registered with the MBMS client to consume DASH Streaming Services, and MBMS client is in `REGISTERED` state for the MAA.

6.3.3.5.4 Expected MBMS Client Actions

When this method is invoked, the MBMS client updates the internal parameters and is expected to provide a `streamingServiceListUpdate()` notification as defined in clause 6.3.3.6. For more details refer to clause 6.3.2.4.

6.3.3.5.5 Usage of Method for MAA

The MAA may invoke the `setStreamingServiceClassFilter()` API to update the previously defined new list of service classes that includes additional service classes or includes fewer service classes than the list of service classes.

The MAA should be aware that the updates are only active once an `streamingServiceListUpdate()` notification is received that confirms the new service class filters.

6.3.3.5.6 Post-Conditions

The MAA expects a `streamingServiceListUpdate()` notification as defined in clause 6.3.3.6.

6.3.3.6 Updating the Streaming Service List

6.3.3.6.1 Overview

This clause defines `streamingServiceListUpdate()` notification.

This notification is used by the MBMS client to inform the MAA about a successful API call `setStreamingServiceClassFilter()` as shown in Figure 6.3.3.5.1-1 or other updates in streaming service list.

6.3.3.6.2 Parameters

None.

6.3.3.6.3 Pre-Conditions

The MBMS client is in `REGISTERED` state for the MAA. The MAA has issued a `setStreamingServiceClassFilter()`API call.

6.3.3.6.4 Expected MBMS Client Actions

The MBMS client issues this notification as a response to a successful `setStreamingServiceClassFilter()`API call or to the response to updates of the service list provided in the MPD. For more details see clause 6.3.2.4.

6.3.3.6.5 Usage of Method for MAA

The MAA is informed about the updates of the service class list and may issues a `getStreamingServices()` API call as defined in clause 6.3.3.4 to obtain the updated service list.

6.3.3.6.6 Post-Conditions

The MAA has the latest service list. No state change is involved.

6.3.3.7 Start DASH Streaming Service

6.3.3.7.1 Overview

This clause defines `startStreamingService()` API.

After the DASH Streaming Application Service registration, the MAA can make calls on the `startStreamingService()` API for the MBMS client to start reception of DASH content received over unicast or broadcast as shown in Figure 6.3.3.7-1.

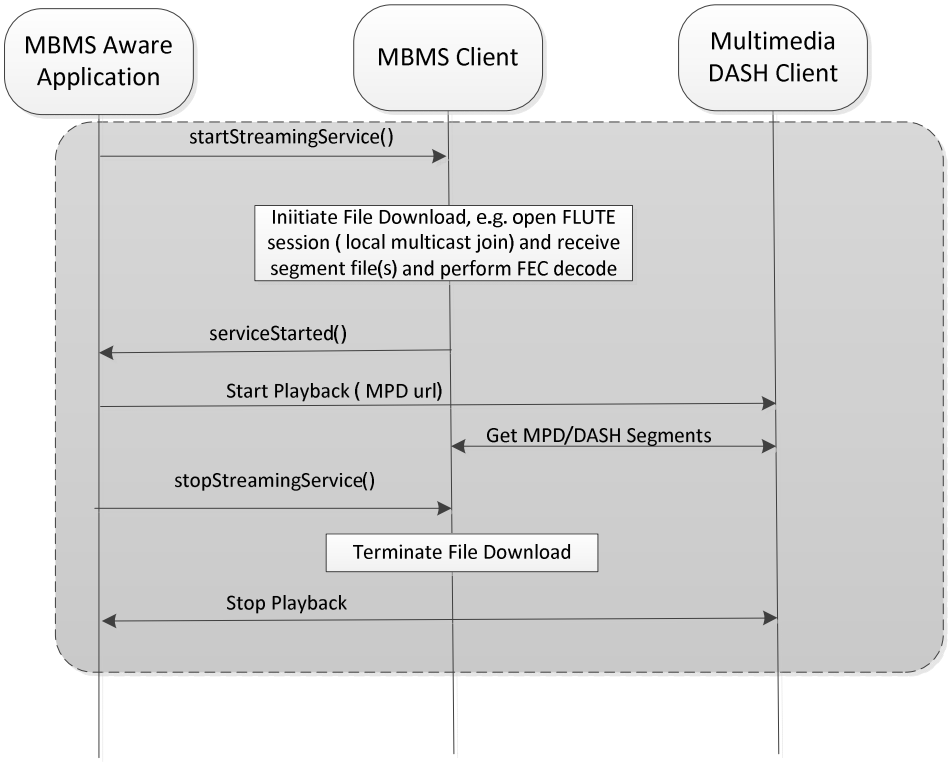

Figure 6.3.3.7-1: MAA starts DASH streaming services

6.3.3.7.2 Parameters

The parameters for the `startStreamingService()` API are:

- `string serviceId` – see definition in clause 6.3.3.2.2.

6.3.3.7.3 Pre-Conditions

The MBMS client is in `REGISTERED` state.

The MAA has the latest service list, for example through the `getStreamingServices()` API call as defined in clause 6.3.3.4.

6.3.3.7.4 Usage of Method for MAA

The MAA can make calls on the `startStreamingService()` API for the MBMS client to start reception of DASH content received over unicast or broadcast.

When MAA is no longer interested in consuming the Streaming Service, it should call the `stopStreaming()` API call as defined in clause 6.3.3.9.

6.3.3.7.5 MBMS Client Actions

When this method is invoked, the MBMS client starts the streaming service, if possible. For more details see clause 6.3.2.4.

6.3.3.7.6 Post-Conditions

When this method is invoked, the MBMS client starts the streaming service, if possible. For more details see clause 6.3.2.4.

6.3.3.8 Notification that DASH Streaming for a Service has started

6.3.3.8.1 Overview

This clause defines `serviceStarted()` callback function.

As illustrated in Figure 6.3.3.7-1, once the MBMS client has successfully collected all necessary information to start the service the MBMS client invokes the `serviceStarted()` callback function.

6.3.3.8.2 Parameters

The parameters for the `serviceStarted()` API are:

- `string serviceId` – see definition in clause 6.3.3.2.2.

6.3.3.8.3 Pre-Conditions

The MAA issued a `startStreamingService()` API call.

The MBMS client is in `REGISTERED` state for the `serviceId`.

6.3.3.8.4 Expected MBMS Client Actions

The MBMS client issues this notification if the service is started successful. For details see clause 6.3.2.4.

6.3.3.8.5 Usage of Method for MAA

Once the MAA receives the callback on the successful start of the service with the `serviceId`, the MAA may start the streaming service by initiating a DASH Media Presentation at a DASH client by handing over the `mpdURI` received during the registration process for this service.

6.3.3.8.6 Post-Conditions

The DASH client can communicate with the MBMS client. The MBMS client makes available the DASH-over-MBMS service based on the MPD referenced in by the `mpdURI` of the service.

6.3.3.9 Stop DASH Streaming Service

6.3.3.9.1 Overview

This clause defines `stopStreamingService()` API.

As Figure 6.3.3.7-1 illustrates, when an MAA that issued a `startStreamingService()` for a service is no longer interested in consuming the DASH content for that service, it will call the `stopStreamingService()` API call.

6.3.3.9.2 Parameters

The parameter for the `stopStreamingService()` API is:

- `string serviceId` – see definition in clause 6.3.3.2.2.

6.3.3.9.3 Pre-Conditions

The MBMS client is in `ACTIVE` state for this MAA.

6.3.3.9.4 Usage of Method for MAA

If an MAA is no longer interested in consuming the DASH service, it should call the `stopStreamingService()` API call. Latest at the same time, the MAA should inform the DASH client about the termination of the service and the DASH client should no longer request resources that are provided directly or referenced by the `mpdURI`.

6.3.3.9.5 MBMS Client Actions

The MBMS terminates the reception. For more details see clause 6.3.2.5.

6.3.3.9.6 Post-Conditions

The MBMS client is in `REGISTERED` state. The Media Presentation referenced by the `mpdURI` can no longer be accessed as the referenced Segments will no longer be provided at the announced location in the MPD.

6.3.3.10 DASH Streaming Application Service De-registration

6.3.3.10.1 Overview

This clause defines `deregisterStreamingApp()` API call.

An MAA registers services classes with the MBMS client to request the start of streaming for DASH Streaming Application Services. The MAA that registered with the MBMS client via the `registerStreamingApp()` API should invoke the `deregisterStreamingApp()` before exiting. The MBMS clients stops monitoring for Service Announcement updates when there are no MAAs registered. There are no parameters for the `registerStreamingApp()` API.

6.3.3.10.2 Parameters

None.

6.3.3.10.3 Pre-Conditions

The MBMS client is in `REGISTERED` state for this MAA.

6.3.3.10.4 Usage of Method for MAA

MAA registered with the MBMS client via the `registerStreamingApp()` API should invoke the `deregisterStreamingApp()` before exiting.

If the MAA deregisters, it will no longer receive notifications from the MBMS client and all context is cleared.

6.3.3.10.5 MBMS Client Actions

The MBMS client no longer sends notifications and clears all context for the MAA.

6.3.3.10.6 Post-Conditions

The MAA is no longer registered with the MBMS client.

The MBMS client is in `IDLE` mode..

6.3.3.11 Notification that DASH Streaming for a Service has stalled

6.3.3.11.1 Overview

This clause defines the `serviceStalled()` notification.

The MBMS client enables consumption of a DASH Streaming Application Service if the current setting for serviceBroadcastAvailability is `BROADCAST_AVAILABLE` or `BROADCAST_UNAVAILABLE`. However, due to UE mobility in and out of broadcast coverage for some DASH Streaming Application Services, the serviceBroadcastAvailability for those services may change to `SERVICE_UNAVAILABLE` (i.e., the UE moves out of coverage for that service). Other circumstances may also prevent the broadcast reception of that service (e.g., a frequency conflict). In these circumstances, the MBMS client can signal the MAA that the service is temporarily not available for playback by invoking the `serviceStalled()` API.

When broadcast reception of the service is re-established, the MBMS client will signal the MAA that the service is again available for playback by invoking the `serviceStarted()` API. This is illustrated in Figure 6.3.3.11.1.

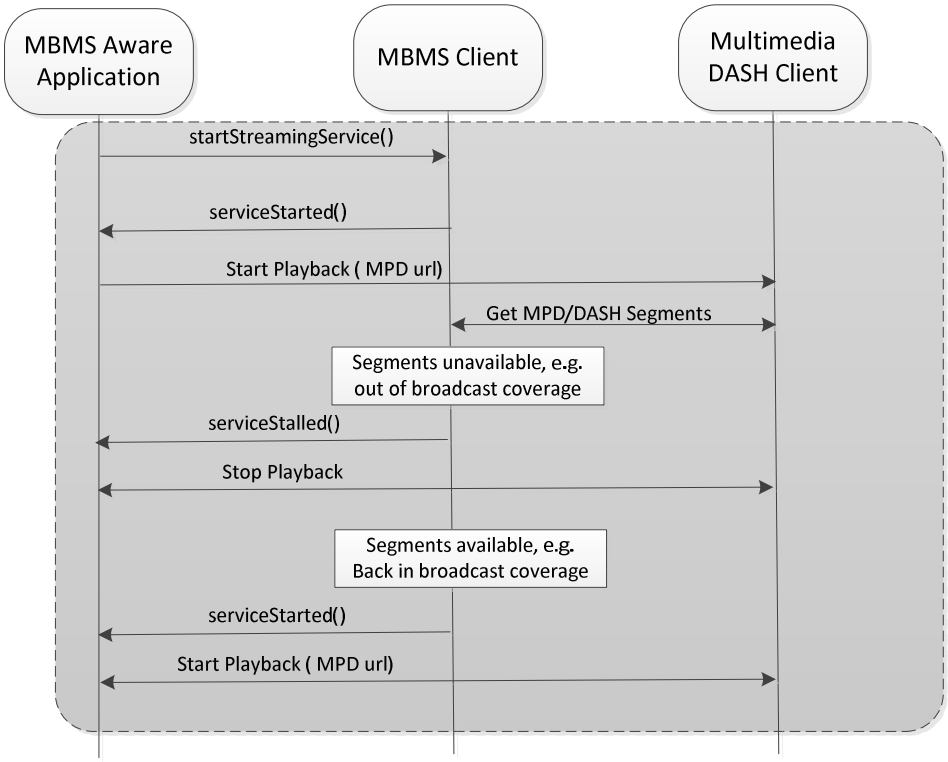

Figure 6.3.3.11.1: Signaling that a DASH streaming service stalled

6.3.3.11.2 Parameters

The parameter for the `serviceStalled()` API are:

- `string serviceId` – identifies the DASH Streaming Application Service for which broadcast receptions have temporarily stalled.
- `StalledReasonCode reason` – provides specific information on what caused the service to stall. Valid options are:
  - `RADIO_CONFLICT` – indicates a frequency conflict, namely the service requested to be started via a `startStreamingService()` cannot be started at this time since the MBMS client is actively receiving another service on a different frequency band.
  - `END_OF_SESSION` – indicates that playback has reached the end of the scheduled transmission for the service as described by the schedule description fragment for the service. This should indicate that the advertised activeServicePeriodEndTime time has been reached.
  - `OUT_OF_COVERAGE` – indicates a UE mobility event to an area where the service with streamingSubtype set to `STREAMING_BC_ONLY` is not available via broadcast.
  - `STALLED_UNKNOWN_REASON` – indicates that another unspecified condition caused the service interruption.

6.3.3.11.3 Pre-Conditions

The MBMS client is in `ACTIVE` mode.

6.3.3.11.4 Expected MBMS Client Actions

The MBMS client provides a `serviceStalled()` notification in case it can no longer provide the referenced resources in the Media Presentation provided with mpdURI. For more details refer to clause 6.3.2.5.

6.3.3.11.5 Usage of Method for MAA

The MAA should stop the DASH client playback on reception of the `serviceStalled()` notification. However, unless the MAA is no longer interested in the content, it should not issue a `stopStreamingService()` call in order to allow the MBMS client from trying to collect DASH content once the download problem is resolved. The MAA should inform the user of the temporary service interruption.

If the DASH client maintains in `STALLED` state for too long, the MAA should stop the service by issuing a `stopStreamingService()`.

6.3.3.11.6 Post-Conditions

The MBMS client is in `STALLED` mode.

6.3.3.12 Notification of DASH Streaming Application Service errors

6.3.3.12.1 Overview

This clause the `streamingServiceError()` notification.

As illustrated in Figure 6.3.3.12.1-1, the `startStreamingService()` request from an MAA may not be served, so the MBMS client will send a failure indication via the `streamingServiceError()` to signal the error code for the result of processing the MAA's `startStreamingService()`.

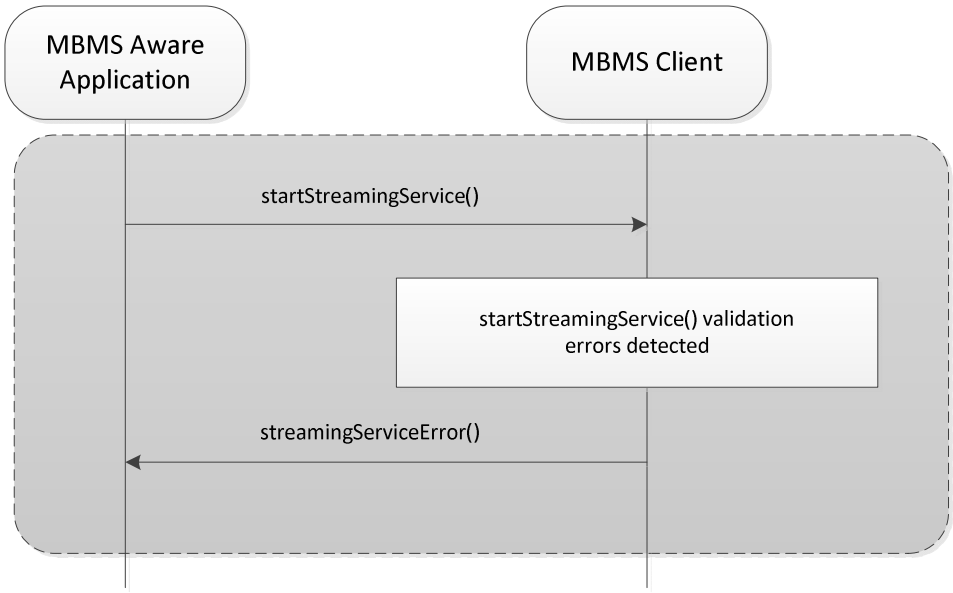

Figure 6.3.3.12.1-1: Signaling errors with the `startStreamingService()` request from the DASH Streaming Application

Figure 6.3.3.12.1-2 also illustrates that the `streamingServiceError()` is used to signal the error code for the result of processing the MAA's a `stopStreamingService()` request.

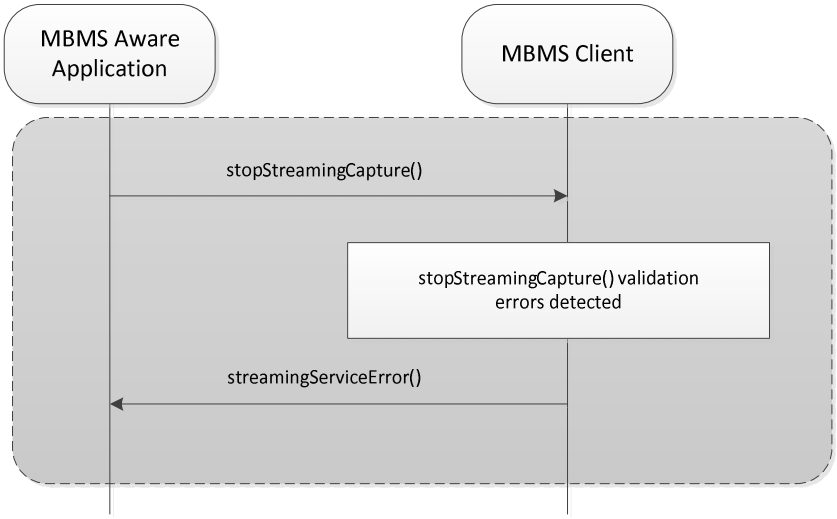

Figure 6.3.3.12.1-2: Signaling errors with the stopStreamingService() request from the DASH Streaming Application

6.3.3.12.2 Parameters

The parameters for the `streamingServiceError()` API are:

- `string serviceId` – identifies the DASH Streaming Application Service on which the MBMS client failed.
- `StreamingErrorCode errorCode` – identifies the error code for the reason causing the `startStreamingService()` or the `stopStreamingService()` request for the serviceId to fail. The available error codes are:
  - `STREAMING_INVALID_SERVICE` – signals that serviceID defined on the `startStreamingService()` or the `stopStreamingService()` request is not currently defined or it is not associated with the service classes with the MAA is registered.
  - `STREAMING_UNKNOWN_ERROR` – signals an error condition not explicitly identified.
- `string errorMsg` – may provide additional textual description of the error condition.

6.3.3.12.3 Pre-Conditions

The MBMS client has received a the `startStreamingService()` or a `stopStreamingService()` request.

6.3.3.12.4 Expected MBMS Client Actions

The MBMS client will send a failure indication via the `streamingServiceError()` to signal the error code for the result of processing the MAA. For more details refer to clause 6.3.2.4 and 6.3.2.5.

6.3.3.12.5 Usage of Method for MAA

If the MAA receives this notification, it should revalidate the capture call. The MAA should also update the service list by issuing a `getStreamingServices()` as defined in clause 6.3.3.4.

6.3.3.12.6 Post-Conditions

No state change is applied.

6.3.3.13 Checking the version for DASH Streaming Application Service interface

6.3.3.13.1 Overview

This clause defines the `getVersion()` request function.

6.3.3.13.2 Parameters

The parameters for the `getVersion()` API call are:

- `string version` – identifies the version of the MBMS clients API implementation.

6.3.3.13.3 Pre-Conditions

The MBMS client may be in any state.

6.3.3.13.4 Usage of Method for MAA

In order for the MAA to know the version of the DASH Streaming Delivery Application Service interface, the `getVersion()` API call may be used. If the version number is not supported by the MAA, it should deregister and not use the API.

6.3.3.13.5 MBMS Client Actions

The `getVersion()` API returns the version of the implemented APIs of the MBMS client.

6.3.3.13.6 Post-Conditions

No state changes apply.

6.4 MBMS Packet Delivery Service API

6.4.1 Introduction

The MBMS packet Streaming delivery Service API provides MAAs with interfaces to manage the reception of packet services delivered over MBMS User services. This API is intended to support packet streaming MAAs.

The IDL for the MBMS Packet Delivery Service API is defined in clause B.4.

The Packet Delivery Service API supports different types of Application Services by using an argument in the service request.

6.4.2 MBMS Client State Model for MBMS packet delivery

6.4.2.1 Overview

Figure 6.4.2.1-1 provides an informative client state model in order to appropriately describe the messages on the MBMS Packet Delivery service API. Four different states are defined as listed in Table 6.4.2.1-1. State changes may happen based on:

- Calls from the MAA or the packet streaming client
- Information provided by the MBMS User Service (USD, schedule, FDT, file complete)
- Changes in the reception conditions

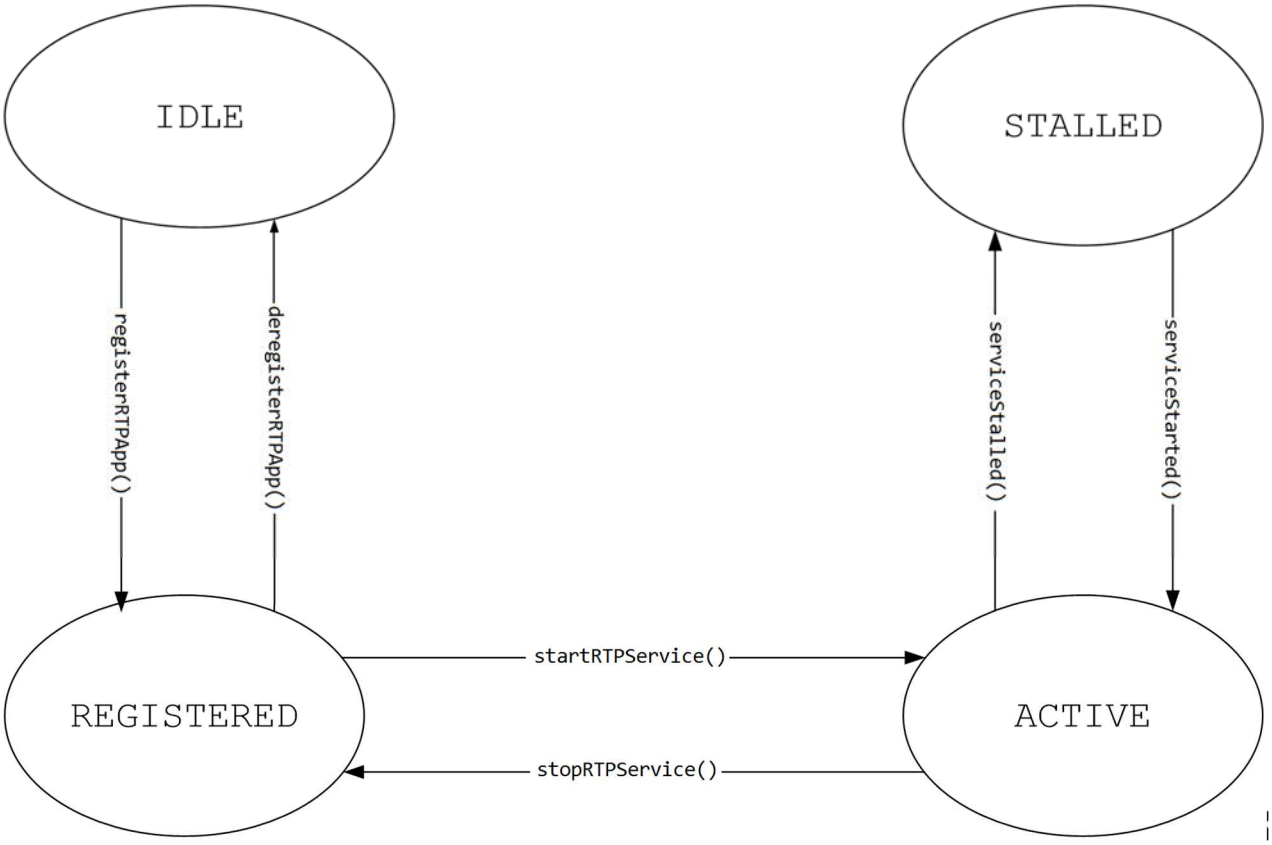

Figure 6.4.2.1-1: State Diagram

Table 6.4.2.1-1 defines states for the MBMS client. Detailed descriptions are provided in the following subclauses.

Table 6.4.2.1-1: States of MBMS Client

States and Parameters	Definition
IDLE	In this state the MBMS client does not have a registered MAA and it may not keep the service definition up to date. For more details see clause 6.4.2.3.
NON_AVAILABLE	In this state the MBMS client is not available and an MAA cannot register with the MBMS client.
REGISTERED	In this state the MBMS client has registered the MAA, it may keep the service definition up to date, and it may be providing file capture services to the MAA(s). In this state the MBMS client sends callback notifications to the MAA. For more details see clause 6.4.2.4.
ACTIVE	In this state the MBMS client provides all services to of the REGISTERED state and also provides the PACKET service to the MAA. For more details see clause 6.4.2.5.
STALLED	In this state the MBMS client provides all services to of the REGISTERED state, but the packet services is at least temporarily stalled. For more details see clause 6.4.2.6.

6.4.2.2 MBMS Client Internal parameters

The MBMS client maintains internal parameters as defined in Table 6.4.2.2-1. Note that the parameters are conceptual and internal and only serve for the purpose to describe message generation on the API calls.

Table 6.4.2.2-1: Parameters of MBMS Client for MBMS Packet Delivery Service

Internal Parameters	Definition
_app[]	The MBMS client maintains a parameter set per registered app
_appId	A unique ID provided by the MAA and assigned to the app.
serviceType	specifies the service type the MAA has requested.
_serviceClass[]	A list of service classes identifying the services the MAA has access to.
_registrationValidityDuration	A period of time following the MAA de-registration over which the MBMS client continues to capture files for the MAA, see clause.
_service[]	The MBMS client maintains a parameter list per service. In this context the list is assigned also to one app, but an implementation may share the internal parameter list assigned to a service across multiple apps.
_serviceID	The service ID for a packet Application service over which the MBMS client collects files for the MAA.
_serviceClass	The service class associated with the packet Application service assigned the Service ID.
_serviceLanguage	The language of the service
_serviceName[] _name _lang	The service name, possibly expressed in different languages.
_serviceBroadcastAvailability	The service broadcast availability for the client. Three different types are defined: BROADCAST_AVAILABLE – UE is in broadcast coverage BROADCAST_UNAVAILABLE – UE is outside of broadcast coverage
_SDP _interfaceName _sdpURI	The latest SDP associated to the service The network interface name from which the started MBMS Packet Delivery service can be received. The URI which is provided to the MAA for initiating the Packet Session.
_sessionSchedule[] _start _stop	Documents the session schedule for this session. Only sessionSchedule records should be included for which the value of the _stop time is in the future.

6.4.2.3 MBMS Client Operation in IDLE state

The MBMS client may listen to the User Service Bundle Description and may collect information. However, no binding with the MAA is in place.

When the registerPacketApp()is invoked, then

1) The MBMS client checks the input parameters for consistency and sets the internal variables:

   a) If the functions of the MBMS client is not accessible, the MBMS client throws a FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE result code in the registerPacketResponse() as defined in subclause 6.4.3.3 and abort the following steps and may at least temporarily move in NOT_AVAILABLE state.

   b) If the MBMS client supports serviceType provided in the service _serviceType record. If the MBMS client does not support the service, it will respond with the appropriate error code, see below.

c) If the service type provided in the `serviceType` field, in the request is recognized by the MBMS client, an empty string then the MBMS client throws a `MISSING_PARAMETER` result code in the `registerPacketResponse()` as defined in subclause 6.4.3.3 and abort the following steps and stays in `IDLE` mode. If not, the MBMS client sets the internal variable `_appId` to the value of the parameter.

d) c) If `appId` is an empty string then the MBMS client throws a `MISSING_PARAMETER` result code in the `registerPacketResponse()` as defined in subclause 6.4.3.3 and abort the following steps and stays in `IDLE` mode. If not, the MBMS client sets the internal variable `_appId` to the value of the parameter.

e) d) The MBMS client adds each entry in the `serviceClassList` parameter to its `_serviceClass[]` record. Note that the `serviceClassList` parameter may contain an empty service class entry. If an empty service class is provided, the MBMS client considers the MAA to be registered with a service class that is also empty and only allow the MAA to have access to MBMS Packet Delivery Application Services that are not associated with a `serviceClass` (i.e., the USD for these services do not have a `serviceClass` defined).

f) e) On receiving a `registerPacketApp()` following a `deregisterPacketApp()`, the MBMS client updates the `serviceClassList` to its `_serviceClass[]` record in the same way described for the `setPacketServiceClassFilter()` method.

g) f) If `callBack` is defined, the MBMS client uses the interfaces in the callback parameter of the **`registerPacketApp()`interface to send notification of event occurrences to the Application.**

2) generates a response `registerPacketResponse()` as defined in subclause 6.4.3.3 and changes to `REGISTERED` state as defined in clause 6.4.2.4:

a) If the MBMS client functions cannot be activated for any reason, especially if the Packet Delivery Application Service API did not find an MBMS client available on the UE on which the MAA is running, the `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE` registration response code is sent. The MBMS client may provide an error message.

b) If the MAA did not provide a mandatory parameter the MBMS client functions cannot be activated, the `MISSING_PARAMETER` registration response code is sent.

c) If the MBMS client does not support the service type, the `NON_SUPPORTED_SERVICE_TYPE` registration response code is sent.

d) If the MBMS client functions can be activated, then:

i) the `RegResponseCode` is set to `REGISTER_SUCCESS` registration response code;

ii) a `message` may be generated.

e) Sends the response with the above parameters

3) If the MBMS client functions can be activated and the response is sent with a `REGISTER_SUCCESS`, then MBMS client is in `REGISTERED` state and uses the `REGISTERED` parameters to provide the list of matching Packet delivery services using the information in the User Service Description (USD). If the response is sent with a `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE`, then MBMS client is in `NOT_AVAILABLE` state. If the response is sent with a `MISSING_PARAMETER` or `NON_SUPPORTED_SERVICE_TYPE` then MBMS client is in `IDLE` state.

If the MBMS client receives the `getVersion()` API call as defined in clause 6.4.3.13, it shall return version 1.0.

6.4.2.4 MBMS Client Operation in REGISTERED state

For each registered MAA and the assigned parameters according to Table 6.4.2.2-1, the MBMS client uses the information in the User Service Description as well as its internal state information for the MAA in `_app[]` using the service type `serviceType` and service class list `_serviceClass[]` to collect and keep up-to-date all internal information for the services of interest for the app, i.e. those that are member of any service class for which the MAA has interest.

For each MBMS user service for which the USD as defined in TS 26.346 [5] is available in the MBMS client for the service type and service classes registered by the MAA in `serviceType` and `_serviceClass[]` and which is

identified as a MBMS Packet Delivery service according to the definition in TS 26.346 [5], clause 8, one service record in the internal parameter _service[] is defined in the MBMS client and continuously updated whenever a new USD is available:

- For each **userServiceDescription.name** element, a (name, lang) pair is generated and added to the _serviceName[] list with _name set to the value of the USD element, and if present, the _lang set to the value of the associated @lang attribute. If no @lang attribute is present, the _lang parameter is set to an empty string.
- If the attribute **userServiceDescription**@serviceClass is present, the value of this attribute is assigned to _serviceClass. If not present, the _serviceClass is set to an empty string.
- The value of the attribute **userServiceDescription**@serviceId is assigned to _serviceId.
- If the attribute **userServiceDescription**@serviceLanguage is present, the value of this attribute is assigned to _serviceLanguage. If not present, the _serviceLanguage is set to an empty string.
- The SDP metadata fragment referenced by either the **r9:sessionDescription** element referencing an SDP and conforming to TS 26.346 [5] is extracted by the MBMS client. The contained SDP is stored in the _SDP parameter. The _sdpURI parameter is generated at which location the SDP will be made available. The name of the network interface from which the Packet data is available is stored in the _interfaceName parameter
- The _serviceBroadcastAvailability is continuously updated set it to BROADCAST_AVAILABLE, if broadcast is available (if the UE is in broadcast coverage of the service), if not, it is set to BROADCAST_UNAVAILABLE (if the UE is NOT in broadcast coverage of the service).
- If the **userServiceDescription.schedule** element is present then the MBMS client uses the information in the schedule description fragment to generate the internal _sessionSchedule[] list and keep up to date as a result of USD updates. The MBMS client shall only include _sessionSchedule[] records if the _stop value is in the future.

If updates are provided and added to the _service[] parameter, the MBMS client should send a packetServiceListUpdate() callback as defined in clause 6.4.3.6.

When the getPacketServices() method is received as defined in clause 6.4.3.4, the MBMS client sets the parameters as follows:

- If the _service[] list is empty, the list is empty
- For each MBMS user service in the service[] list, one service record is generated as follows:
  - The value of the attribute _serviceId is assigned to serviceId.
  - The value of the attribute _serviceClass is assigned to serviceClass.
  - The value of _serviceLanguage is assigned to serviceLanguage.
  - For each record in the _serviceName[] one serviceNameList entry is generated and:
    - the name is set to the value _name,
    - the lang is set to the value _lang.
  - The value of _serviceBroadcastAvailability is assigned to serviceBroadcastAvailability.
  - The _sdpURI is assigned to spdURI.
  - If at least one _sessionSchedule[] record is present then
    - The activeServicePeriodStartTime is set to the value of earliest _start time of any entry in the _sessionSchedule[].

- The `activeServicePeriodStopTime` is set to the value of the `_stop` time of the entry selected earliest start time.

- If no `_sessionSchedule[]` record is present:

  - The `activeServicePeriodStartTime` is set to 0.

  - The `activeServicePeriodStopTime` is set to 0.

When the `setPacketServiceClassFilter()` as defined in clause 6.4.3.5 is received, the MBMS client runs the following steps:

- It replaces the internal variable `_serviceClass[]` with the parameter values provided in `serviceClassList`.

- The MBMS client dis-associates the service classes previously associated with the MAA that are not included on this list.

- The MBMS client associates the service classes not previously associated with the MAA that are newly included on this list.

- The MBMS client issues a `rtpServiceListUpdate()` notification as defined in clause 6.4.3.6 to the MAA to notify of this effect.

When the `startPacketService()` method as defined in clause 6.4.3.7 is received with a parameter `serviceID`, the MBMS client runs the following steps:

- The MBMS client checks for errors and if necessary, the `packetServiceError()` notification as defined in clause 6.4.3.12 is initiated. Specifically, if the MBMS client does not find a matching `serviceId` in its internal `_service[]` record, it responds with error code `PACKET_INVALID_SERVICE`. Otherwise it may use the error code `PACKET_UNKNOWN_ERROR`. An errorMsg may be provided in the `errorMsg` string.

- If the service with the `serviceId` parameter can be started:

  - The MBMS client uses the SDP in the `_SDP` parameter and the remaining associated metadata to offer a valid session to the packet client by providing a server in the MBMS client. Clause 7 provides different provides service specific interfaces.

  - The URL to the SDP that is exposed to the MAA for Packet consumption is stored in the internal variable `_sdpURI`. The SDP stored at this URI shall not be changed for a session.

  - The MBMS client sends a `serviceStarted()` notification as defined in clause 6.4.3.8 with the `serviceId` being passed along with the notification.

  - The MBMS client moves to `ACTIVE` state as defined in clause 6.4.2.5.

Whenever there has been a change to the parameters reported to the MAA in response to a `getPacketServices()` API, i.e. in the internal service class list `_serviceClass[]` to add a new service record to the list or a change in one of the following internal parameters in the service record in the `_serviceLanguage`, `_serviceName[]_serviceBroadcastAvailability`, or updates to the `_sdpURI` the MBMS client notifies MAA with `packetServiceListUpdate()` as defined in clause 6.4.3.6.

When the `deregisterPacketApp()` is received, all internal parameters for the MAA are cleared and the client moves to `IDLE` state.

6.4.2.5 MBMS Client Operation in ACTIVE state

The MBMS client carries out all actions as in the `REGISTERED` state.

The MBMS client continuously receives the packet data and makes it available as announced in the SDP. For different options depending on the service type, refer to clause 7. The URL to the SDP that is exposed to the MAA for Packet consumption is stored in the internal variable `_sdpURI`.

When the MBMS client receives a `stopPacketService()` request as defined in clause 6.4.3.9 that matches an active service, then

- the MBMS client checks for errors and if necessary, the `packetServiceError()` notification as defined in clause 6.4.3.12 is initiated. Specifically, if the MBMS client does not find a matching `serviceId` in its internal `_service[]` record, it responds with error code `PACKET_INVALID_SERVICE`. Otherwise it may use the error code `PACKET_UNKNOWN_ERROR`. An errorMsg may be provided in the `errorMsg` string.
- the MBMS client stops providing the session at its internal server, i.e. at the location announced in the SDP referenced by the `_sdpURI`.
- The MBMS client moves to `REGISTERED` state as defined in clause 6.4.2.4.

When the MBMS client receives a `stopPacketService()` request as defined in clause 6.4.3.9 that matches an active service, the MBMS client terminates the reception of the data of this delivery session and no longer makes it available at the indicated resources in the SDP. The MBMS client transitions to `REGISTERED` state

When the MBMS the internal parameter `_serviceBroadcastAvailability` transitions to `BROADCAST_UNAVAILABLE`, and no alternative delivery method is defined, or if the service is no longer available for other reasons (e.g. frequency conflict), then the service is stalled. In this case the MBMS client:

- no longer makes available the resources in the announced locations by the `_sdpURI` and the references therein,
- sends a `serviceStalled()` notification as defined in clause 6.4.3.11, along with one of the following reasons:
  - `RADIO_CONFLICT` – indicates a frequency conflict, namely the service requested to be started via a `startPacketService()` cannot be started at this time since the MBMS client is actively receiving another service on a different frequency band.
  - `END_OF_SESSION` – indicates that playback has reached the end of the scheduled transmission for the service as described by the schedule description fragment for the service. This should indicate that the advertised activeServicePeriodEndTime time has been reached.
  - `OUT_OF_COVERAGE` – indicates a UE mobility event to an area where the service is not available via broadcast.
  - `STALLED_UNKNOWN_REASON` – indicates that another unspecified condition caused the service interruption.
- Transitions to the `STALLED` state as defined in clause 6.4.2.6.

6.4.2.6 MBMS Client Operation in STALLED state

The MBMS client carries out all actions as in the `REGISTERED` state.

In this state the MBMS client continuously monitors if the service can be made available again.

Once the service gets available again, the MBMS client

- The MBMS client receives the packet streams and makes them available as announced in the SDP. The URL to the SDP that is exposed to the MAA for Packet consumption is stored in the internal variable `_sdpURI`.
- The MBMS client sends a `serviceStarted()` notification as defined in clause 6.4.3.8 with the `serviceId` being passed along with the notification.
- The MBMS client moves to `ACTIVE` state as defined in clause 6.4.2.5.

6.4.3 Methods

6.4.3.1 Overview

Table 6.4-2 provides an overview over the methods defined for the MBMS Packet Delivery Service API. Different types are provided, namely state changes triggered by the MAA, status query of the MAA to the client, parameter updates as well as notifications from the client. The direction of the main communication flow is provided between the MAA (A) and the MBMS Client (C).

Table 6.4-2: Methods defined for MBMS Packet Delivery Service API

Method	Type	Direction	Brief Description	Section
`registerPacketApp`	State change	A -> C	MAA registers a callback listener with the MBMS client	6.4.3.2
`deregisterPacketApp`	State change	A -> C	MAA deregisters with the MBMS client	6.4.3.10
`startPacketService`	State change	A -> C	Starts Packet service	6.4.3.7
`stopPacketService`	State change	A -> C	Stop Packet service	6.4.3.9
`getPacketServices`	Status query	C <-> A	Get list of currently active services	6.4.3.4
`getVersion`	Status query	C <-> A	Get the API version	6.4.3.13
`setPacketServiceClassFilter`	Update to parameter list	A -> C	MAA sets a filter on MBMS Packet Delivery services in which it is interested	6.4.3.5
`registerPacketResponse`	Update to parameter list	C -> A	The response to the MAA Packet service register API	6.4.3.3
`serviceStarted`	Notification	C -> A	Notification to MAA when the MBMS Packet Delivery service started	6.4.3.8
`rtpServiceListUpdate`	Notification	C -> A	Notification to MAA on an update of the available for MBMS Packet Delivery services	6.4.3.6
`rtpServiceError`	Notification	C -> A	Notification to MAA when there is an error with the reception download of service	6.4.3.12
`serviceStalled`	Notification	C -> A	Notification to MAA that MBMS Packet Delivery service failed	6.4.3.11

6.4.3.2 Registration

6.4.3.2.1 Overview

This clause defines `registerPacketApp()` interface.

An MAA calls the `registerPacketApp()` interface to register with the MBMS Client to consume MBMS Packet Delivery services. The `registerPacketApp()` interface has two purposes:

- It signals to the MBMS Client that an MAA is interested to consume MBMS Packet Delivery Services.
- It allows the MAA to identify its callback listeners defined in the MBMS Packet Delivery service API for the MBMS Client to provide asynchronous notifications to the MAA on relevant events associated with Packet-over-MBMS.

Note: Since some application development frameworks do not support callback functions, an MAA for these frameworks will not provide callback listeners in the `registerPacketApp()` interface. Instead, the MAA will implement the necessary approach available on these frameworks to receive event notifications from the MBMS Client in place of callback functions. The notifications implemented on these frameworks will include the same information content as defined on the structures for the IDL callback functions.

Figure 6.4-2 shows the registration process.

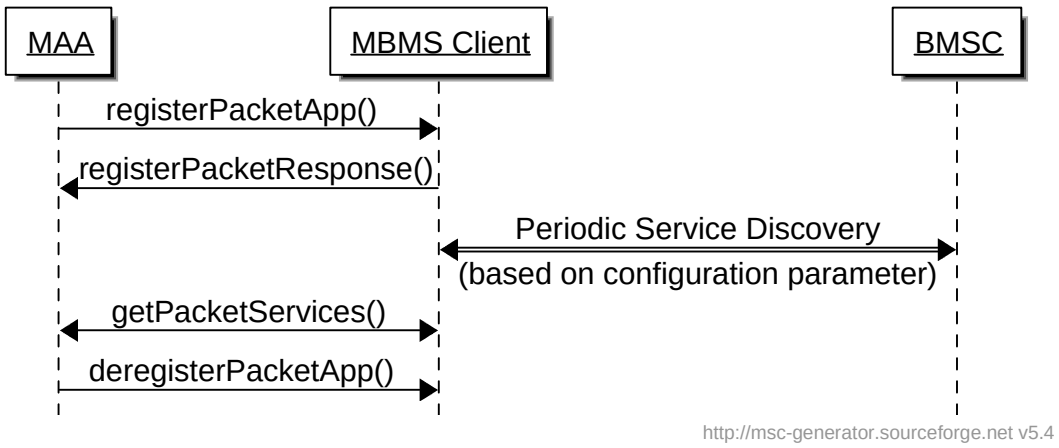

Figure 6.4-2 MAA Registration sequence diagram

6.4.3.2.2 Parameters

The parameters for the `registerPacketApp()` API are:

- **`ServiceType`** `serviceType` – provides the service type that the MAA wants to access. Different service types for packet streaming are defined in clause 4.3.
- **`string`** `appId` – provides a unique ID for the MAA registering with the MBMS client, which uses this identity to maintain state information for a particular MAA. The uniqueness of the ID is in the context of any MAA that may possibly register with MBMS client. Uniqueness is typically provided on platform level.
- **`any`** `platformSpecificAppContext` – a platform-specific context for the registering MAA that enables the MBMS client to get extra information about the MAA that may be need to enable the MAA to have access to MBMS services, e.g., to enable MAA authentication or to enable the MAA to communicate with the MBMS client via platform (e.g., HLOS) services.
- **`sequence<string>`** `serviceClassList` – provides a comma-separated list of service classes which the MAA is interested to register. Each service class string can be any string or it may be empty.
- **`ILTEPacketServiceCallback`** `callBack` – provides the MBMS client with the call back functions associated with MBMS Packet Delivery Service APIs for the registering MAA.

NOTE: The callback element in the IDL description is optional and only included when the MAA development framework supports programmatic callback interfaces. If callbacks are not supported on a given MAA development framework, the same information content as defined on the callback structures is to be provided to the MAA via the notification method available with that development framework when the respective condition is met.

6.4.3.2.3 Pre-Conditions

The MAA has assigned a unique application ID `appId` in the context of its operation (e.g., a smartphone HLOS) with the MBMS client.

The MAA is pre-configured with the set of service classes that allows it to consume the MBMS packet delivery services of the specified type associated with these service classes.

The MAA has access to a packet client that can consume the service type.

The MAA may use this method at launch or after a `deregisterPacketApp()` has been called.

The MBMS client is in `IDLE` state.

6.4.3.2.4 Usage of Method for MAA

The MAA uses the method `registerPacketApp()` to register with the MBMS Client to consume Packet Delivery services of a specific type using the `serviceType` parameter. Different service types are defined in clause 4.3.

The MAA provides its `appId` and, if applicable, some platform specific MAA context, `platformSpecificAppContext`.

The MAA provides the set of service classes which the MAA is interested to register.

6.4.3.2.5 Expected MBMS Client Actions

When this method is invoked, the MBMS registers the MAA, if possible. For more details refer to clause 6.4.2.3.

6.4.3.2.6 Post-Conditions

The MAA expects the `registerPacketResponse()` as defined in 6.4.3.3.

6.4.3.3 MBMS Packet Delivery Service Registration Response

6.4.3.3.1 Overview

This clause defines `registerPacketResponse()` call.

As illustrated in Figure 6.4-2, the MBMS client responds to an MAA call to the `registerPacketApp()` API with a `registerPacketResponse()` call back providing the result of the registration request.

6.4.3.3.2 Parameters

The parameters for the `registerPacketResponse()` API are:

- `EmbmsCommonTypes::RegResponseCode` value – provides a result code on the registration request. The allowed values are:
  - `REGISTER_SUCCESS` – indicates that the registration has been processed successfully and the MAA can proceed with other API interactions with the MBMS client for MBMS Packet Delivery Services.
  - `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE` – Indicates that the registration has failed since the MBMS Packet Delivery Service API did not find an MBMS client available on the UE on which the MAA is running and no MBMS service will be available to the MAA.
  - `MISSING_PARAMETER` – indicates that the registration has failed since one or more of the required parameter was missing.
  - `NOT_SUPPORTED_SERVICE_TYPE` – indicates that the service type requested in `serviceType` is not supported by the MBMS client.
- `string message` – provides an associated text description of the error message. The message may be empty.

6.4.3.3.3 Pre-Conditions

The MBMS client has received a call via the `registerPacketApp()` API as defined in clause 6.4.3.2.

6.4.3.3.4 Expected MBMS Client Actions

The MBMS client responds accordingly and depending on the response moves to one of the states: `IDLE`, `NOT_AVAILABLE`, or `REGISTERED`. For more details refer to clause 6.4.2.4.

6.4.3.3.5 Usage of Method for MAA

Once the MAA receives a the `registerPacketResponse()` with the `RegResponseCode` set to `REGISTER_SUCCESS`, the MAA can proceed with other API interactions with the MBMS client.

If the MBMS client is temporarily in `NOT_AVAILABLE`, if the `registerPacketResponse()` signaled a failure with a `FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE`, the MAA may periodically recheck if the state of the MBMS client changes by retrying the `registerPacketApp()` API.

If the MBMS client is responding with `MISSING_PARAMETERS`, the MAA should fix the parameters and retry the `registerPacketApp()` API.

If the MBMS client is responding with `NON_SUPPORTED_SERVICE_TYPE`, the MAA may check if other service types can be used to possible access the services, for example if the MAA supports handling other service types.

6.4.3.3.6 Post-Conditions

If the MBMS client functions cannot be activated and once the response is sent, then MBMS client is at least temporarily in `NOT_AVAILABLE` state.

If the MBMS client functions can be activated and respective response is sent, then MBMS client is in `REGISTERED` state with the `REGISTERED` parameters as set above.

6.4.3.4 Getting information on available MBMS Packet Delivery Services

6.4.3.4.1 Overview

This clause defines `getPacketServices()` API call.

The `registerPacketApp()` interface returns the complete list of available MBMS Packet Delivery Services information. As illustrated in Figure 6.4-2, after a successful registration with the MBMS client, the MAA can use the `getPacketServices()` API to discover the available MBMS Packet Delivery Services associated with the service classes registered via the `registerPacketApp()`.

6.4.3.4.2 Parameters

The `getPacketServices()` API returns a list describing the available MBMS Packet Delivery Services, where each service is described by the following output only parameters:

- `sequence<ServiceNameLang> serviceNameList` – optionally provides a list of the service title name in possibly different languages. Each (name, lang) pair defines a title for the service on the language indicated.
  - `string name` – offers a title for the user service on the language identified in the lang parameter.
  - `string lang` – identifies a natural language identifier per RFC 3066 [10].
- `string serviceClass` – identifies the service class which is associated with the service.
- `string serviceId` – provides the unique service ID for the service. The uniqueness is among all services provided by the BMSC.
- `string serviceLanguage` – indicates the available language for the service and represented as an identifier per RFC3066 [10].
- `EmbmsCommonTypes::ServiceAvailabilityType serviceBroadcastAvailability` – signals whether the UE is currently in the broadcast coverage area for the service.
  - The possible values are:
    - `BROADCAST_AVAILABLE` – if content for the service is broadcast at the current device location.
    - `BROADCAST_UNAVAILABLE` – if content for the service is not broadcast at the current device location.
- `string sdpUri` – provides an HTTP URL where the SDP for the MBMS Packet Delivery Service is hosted and available for media client access.
- `string interfaceName` – provides the network interface name from which the started MBMS Packet Delivery Service can be received by the media clients.
- `EmbmsCommonTypes::Date activeServicePeriodStartTime` – signals the current/next active MBMS Packet Delivery Service start time, when data starts being broadcast over the air.

- `EmbmsCommonTypes::Date activeServicePeriodEndTime` – signals the current/next active MBMS Packet Delivery Service stop time, when Packet data stops being broadcasted over the air.

6.4.3.4.3 Pre-Conditions

The MBMS client is in `REGISTERED` state.

6.4.3.4.4 Expected MBMS Client Actions

When this method is invoked, the MBMS client returns the Packet service parameters. For more details refer to clause 6.4.2.4.

6.4.3.4.5 Usage of Method for MAA

The MAA should use this call right after the `registerPacketResponse()` notification as defined in clause 6.4.3.3 is received or after the `rtpServiceListUpdate()` notification as defined in clause 6.4.3.6 is received.

The MAA should use the `serviceId` to identify the service in subsequent communication with the MBMS client to manage the MBMS Packet Delivery service.

The usage of the parameters `serviceNameList`, `serviceClass`, `serviceBroadcastAvailability`, and `serviceLanguage` is typically up to the MAA.

The `sdpURI` should be used by the MAA to initiate playback by initiating a Packet client. The MAA should assume that the Packet Session can be consumed by the Packet client without any further interaction with the MAA. The `interfaceName` should be used by the Packet client to filter the data stream reception from a specific network interface.

The parameters `activeServicePeriodStartTime` and `activeServicePeriodEndTime` provides the MAA the ability to determine the current broadcast state for the service as follows:

- If the current time is such that `activeServicePeriodStartTime` ≤ `current time` ≤ `activeServicePeriodEndTime`, Packet content is being broadcast for the service at the current time.
- If the `activeServicePeriodStartTime` is in the future, there is currently no broadcast being made for the service, but broadcast transmission is currently scheduled to start at this advertised time.
- If the `activeServicePeriodStartTime` is set to zero, there is no currently defined broadcast schedule time for the service.

6.4.3.4.6 Post-Conditions

This call does not change the MBMS client state.

The MAA uses the `serviceId` to identify the service in subsequent communication with the MBMS client.

6.4.3.5 Updating the registered service classes

6.4.3.5.1 Overview

This clause defines `setPacketServiceClassFilter()` call.

While an MAA is actively registered with the MBMS client to consume MBMS Packet Delivery services, the MAA can call the `setPacketServiceClassFilter()` API to update the list of service classes the MAA wants to be registered with, see figure 6.4.3.5.1-1.

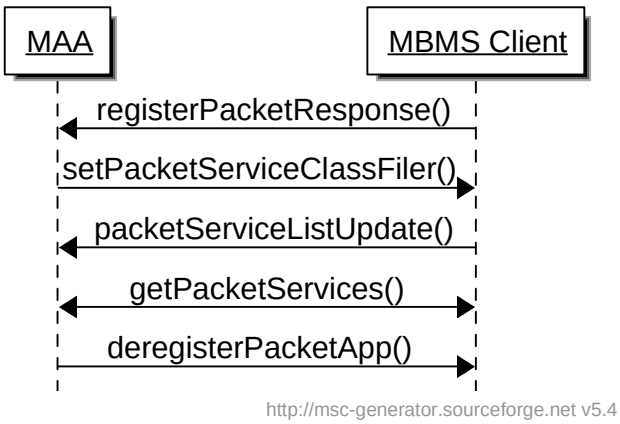

Figure 6.4.3.5.1-1: Sequence diagram for updating the registered service classes for an MAA

6.4.3.5.2 Parameters

The parameters for the `setPacketServiceClassFilter()` method are:

- sequence<string> serviceClassList – see clause 6.4.3.2.2.

6.4.3.5.3 Pre-Conditions

The MAA is actively registered with the MBMS client to consume MBMS Packet Delivery services, and MBMS client is in `REGISTERED` state for the MAA.

6.4.3.5.4 Expected MBMS Client Actions

When this method is invoked, the MBMS client shall update the internal parameters and is expected to provide a `packetServiceListUpdate()` notification as defined in clause 6.4.3.6. For more details refer to clause 6.4.2.4.

6.4.3.5.5 Usage of Method for MAA

The MAA may invoke the `setPacketServiceClassFilter()` API to update the previously defined new list of service classes that includes additional service classes or includes fewer service classes than the list of service classes.

The MAA should be aware that the updates are only active once an `packetServiceListUpdate()` notification is received that confirms the new service class filters.

6.4.3.5.6 Post-Conditions

The MBMS client issues an `packetServiceListUpdate()` notification as defined in clause 6.4.3.6.

6.4.3.6 Updating the Packet Service List

6.4.3.6.1 Overview

This clause defines `packetServiceListUpdate()` notification.

This notification is used by the MBMS client to inform the MAA about a successful API call `setPacketServiceClassFilter()` as shown in Figure 6.4-3 or other updates in Packet service list.

6.4.3.6.2 Parameters

None.

6.4.3.6.3 Pre-Conditions

The MBMS client is in `REGISTERED` state for the MAA. The MAA has issued a `setPacketServiceClassFilter()`API call.

6.3.3.6.4 Expected MBMS Client Actions

The MBMS client issues this notification as a response to a successful `setPacketServiceClassFilter()`API call or to the response to updates of the service list provided in the SDP. For more details see clause 6.4.2.4.

6.4.3.6.5 Usage of Method for MAA

The MAA is informed about the updates of the service class list and may issues a `getPacketServices()` API call as defined in clause 6.4.3.4 to obtain the updated service list.

6.4.3.6.6 Post-Conditions

The MAA has the latest service list. No state change is involved.

6.4.3.7 Start MBMS Packet Delivery Service

6.4.3.7.1 Overview

This clause defines `startPacketService()` API.

After the MBMS Packet Delivery Service registration, the MAA can make calls on the `startPacketService()` API for the MBMS client to start reception of content received over broadcast, as shown in Figure 6.4.3.7.1-1.

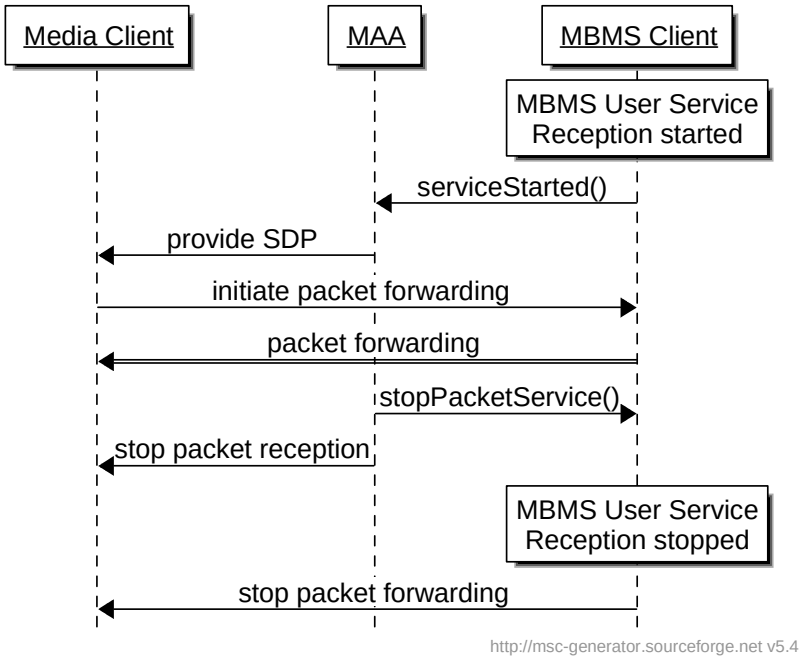

Figure 6.4.3.7.1-1: MAA starts MBMS Packet Delivery services

6.4.3.7.2 Parameters

The parameters for the `startPacketService()` API are:

- `string serviceId` – see clause 6.4.3.4.2.

6.4.3.7.3 Pre-Conditions

The MBMS client is in `REGISTERED` state.

The MAA has the latest service list, for example through the `getPacketServices()` API call as defined in clause 6.4.3.4.

6.4.3.7.4 Usage of Method for MAA

The MAA can make calls on the `startPacketService()` API for the MBMS client to start reception of Packet data over the MBMS system.

When the MAA is no longer interested in consuming the Packet Session, it should call the `stopPacketService()` API call as defined in clause 6.4.3.9. It should also inform the media client to stop packet reception.

6.4.3.7.5 MBMS Client Actions

When this method is invoked, the MBMS client starts the Packet service, if possible. For more details see clause 6.4.2.4.

6.4.3.7.6 Post-Conditions

The MAA expects a `serviceStarted()` notification as defined in clause 6.4.3.8 or an appropriate error message.

6.4.3.8 Notification that MBMS Packet Delivery Service has started

6.4.3.8.1 Overview

Once the MBMS client has successfully collected all necessary information to start the service the MBMS client invokes the `serviceStarted()` callback function.

6.4.3.8.2 Parameters

The parameters for the `serviceStarted()` API are:

- `string serviceId` – see definition in clause 6.4.3.2.2.

6.4.3.8.3 Pre-Conditions

The MAA issued a `startPacketService()` API call.

The MBMS client is in `REGISTERED` state for the `serviceId`.

6.4.3.8.4 Expected MBMS Client Actions

The MBMS client issues this notification if the service is started successful. For details see clause 6.4.2.4.

6.4.3.8.5 Usage of Method for MAA

Once the MAA receives the callback on the successful start of the service with the `serviceId`, the MAA may start the MBMS Packet Delivery service initiating a Packet Session at a Packet client by handing over the `sdpURI` received during the registration process for this service. The media client iniates the packet forwarding.

6.4.3.8.6 Post-Conditions

The Packet client can communicate with the MBMS client. The MBMS client makes available the MBMS Packet Delivery service based on the SDP referenced in by the `sdpURI` of the service.

6.4.3.9 Stop MBMS Packet Delivery Service

6.4.3.9.1 Overview

This clause defines `stopPacketService()` API.

As figure 6.4-4 illustrates, when an MAA that issued a `startPacketService()` for a service is no longer interested in consuming the Packet session for that service, it will call the `stopPacketService()` API call.

6.4.3.9.2 Parameters

The parameter for the `stopPacketService()` API is:

- `string serviceId` – see definition in clause 6.4.3.2.2.

6.4.3.9.3 Pre-Conditions

The MBMS client is in `ACTIVE` state for this MAA.

6.4.3.9.4 Usage of Method for MAA

If an MAA is no longer interested in consuming the Packet service, it should call the `stopPacketService()` API call. Latest at the same time, the MAA should inform the Packet client about the termination of the service and the Packet client should no longer receive data that are referenced by the `sdpURI`.

6.4.3.9.5 MBMS Client Actions

The MBMS terminates the reception and no longer forwards the packets to the media client. For more details see clause 6.4.2.5.

6.4.3.9.6 Post-Conditions

The MBMS client is in `REGISTERED` state. The Packet Session referenced by the `sdpURI` can no longer be accessed as the referenced data will no longer be provided at the announced location in the SDP.

6.4.3.10 MBMS Packet Delivery Service De-registration

6.4.3.10.1 Overview

This clause defines `deregisterPacketApp()` API.

An MAA registers service classes with the MBMS client to request the start of streaming for MBMS Packet Delivery Services. The MAA that registered with the MBMS client via the `registerPacketApp()` API should invoke the `deregisterPacketApp()` before exiting. An implicit `stopPacketService()` call is performed for all MBMS Packet Delivery Services that have been started since the last `registerPacketApp()` call. If there are no MAA interested in an MBMS Packet Delivery Service, the MBMS client stops capturing data for this Service.

The MBMS client stops monitoring for Service Announcement updates when there are no MAAs registered. There are no parameters for the `registerPacketApp()` API.

6.4.3.10.2 Parameters

None.

6.4.3.10.3 Pre-Conditions

The MBMS client is in `REGISTERED` state for this MAA.

6.4.3.10.4 Usage of Method for MAA

MAA registered with the MBMS client via the `registerPacketApp()` API should invoke the `deregisterPacketApp()` before exiting.

6.4.3.10.5 MBMS Client Actions

The MBMS client no longer sends notifications and clears all context for the MAA.

6.4.3.10.6 Post-Conditions

The MAA is no longer registered with the MBMS client.

The MBMS client is in `IDLE` mode.

6.4.3.11 Notification that MBMS Packet Delivery Service has stalled

6.4.3.11.1 Overview

This clause the `serviceStalled()` notification.

The MBMS client enables consumption of a MBMS Packet Delivery service if the current setting for serviceBroadcastAvailability is `BROADCAST_AVAILABLE` or `BROADCAST_UNAVAILABLE`. Other circumstances may also prevent the broadcast reception of that service (e.g., a frequency conflict). In these circumstances, the MBMS client will signal the MAA that the service is temporarily not available for playback by invoking the `serviceStalled()` API.

When broadcast reception of the service is re-established, the MBMS client will signal the MAA that the service is again available for playback by invoking the `serviceStarted()` API. This is illustrated in Figure 6.4-5.

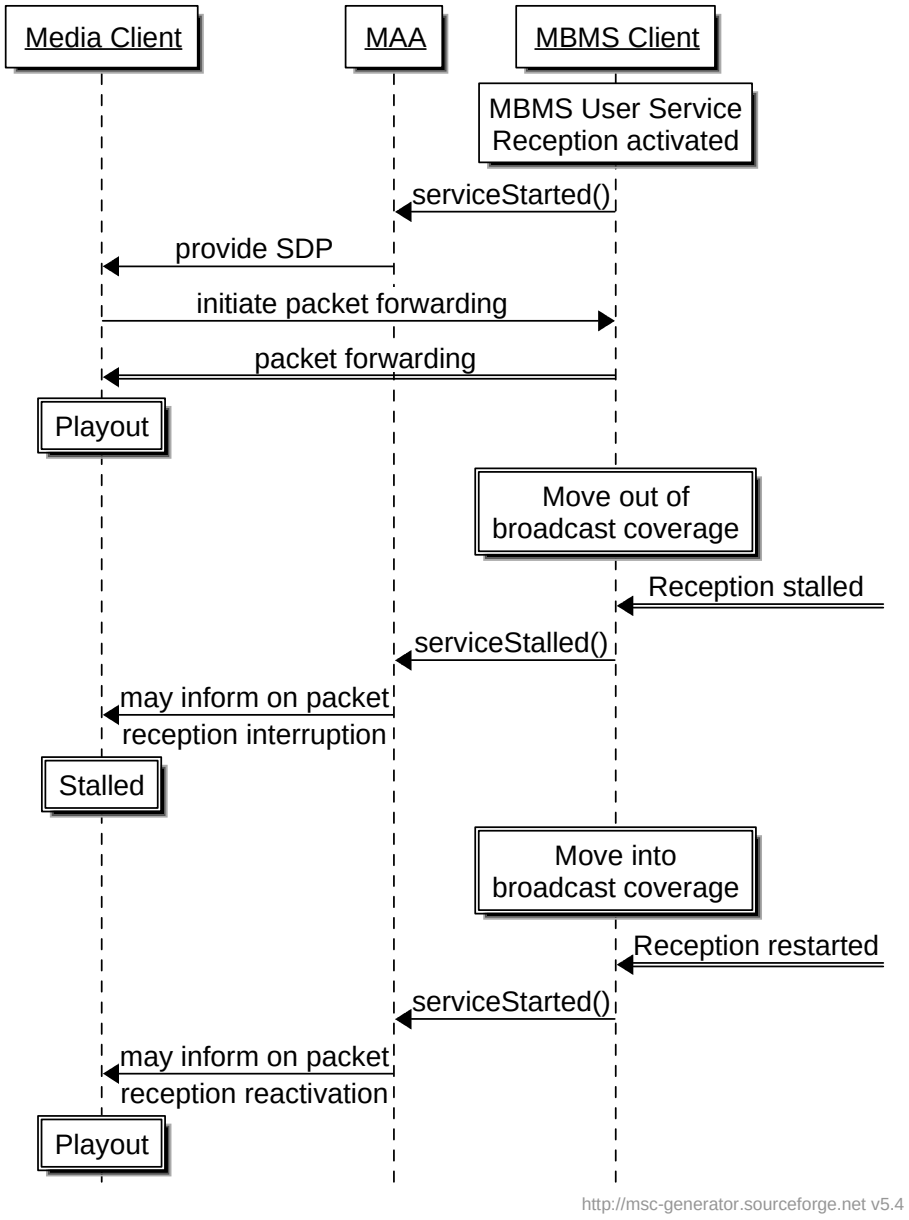

Figure 6.4-5: Signaling that a MBMS Packet Delivery service stalled

6.4.3.11.2 Parameters

The parameter for the `serviceStalled()` API are:

- `string serviceId` – identifies the MBMS Packet Delivery Service for which broadcast receptions have temporarily stalled.
- `StalledReasonCode reason` – provides specific information on what caused the service to stall. Valid options are:

- RADIO_CONFLICT – indicates a frequency conflict, namely the service requested to be started via a startPacketService() cannot be started at this time since the MBMS client is actively receiving another service on a different frequency band.
- END_OF_SESSION – indicates that playback has reached the end of the scheduled transmission for the service as described by the schedule description fragment for the service. This should indicate that the advertised activeServicePeriodEndTime time has been reached.
- OUT_OF_COVERAGE – indicates a UE mobility event to an area where the service is not available via broadcast.

- STALLED_UNKNOWN_REASON – indicates that another unspecified condition caused the service interruption.

6.4.3.11.3 Pre-Conditions

The MBMS client is in ACTIVE mode.

6.4.3.11.4 Expected MBMS Client Actions

The MBMS client provides a serviceStalled() notification in case it can no longer provide the referenced resources in the Packet Session provided with sdpURI. For more details refer to clause 6.4.2.5.

6.4.3.11.5 Usage of Method for MAA

The MAA should stop the Packet client playback on reception of the serviceStalled() notification. However, unless the MAA is no longer interested in the content, it should not issue a stopPacketService() call in order to allow the MBMS client from trying to collect Packet content once the download problem is resolved. The MAA may inform the user of the temporary service interruption.

If the media client maintains in STALLED state for too long, the MAA should stop the service by issuing a stopPacketService().

6.4.3.11.6 Post-Conditions

The MBMS client is in STALLED mode.

6.4.3.12 Notification of MBMS Packet Delivery Service errors

6.4.3.12.1 Overview

This clause defines the packetServiceError() notification.

As illustrated in Figure 6.4-5, the startPacketService() request from an MAA may not be served, so the MBMS client will send a failure indication via the packetServiceError() to signal the error code for the result of processing the MAA's startPacketService().

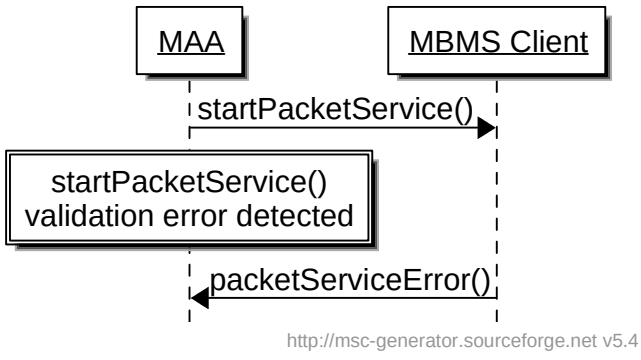

Figure 6.4-5: Signaling errors with the startPacketService() request from the Packet-over-MBMS

Figure 6.4-6 also illustrates that the packetServiceError() is used to signal the error code for the result of processing the MAA's a stopPacketService() request.

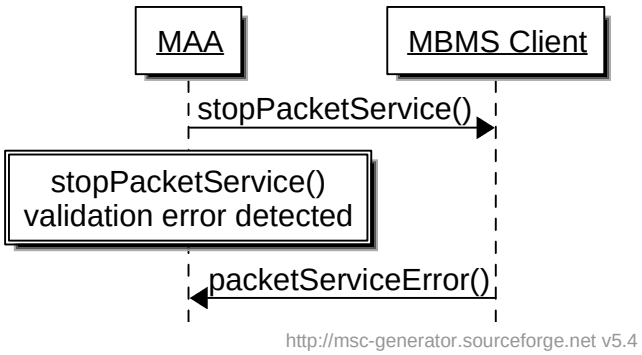

Figure 6.4-6: Signaling errors with the stopPacketService() request from the Packet-over-MBMS

6.4.3.12.2 Parameters

The parameters for the `rtpServiceError()` API are:

- `string serviceId` – identifies the MBMS Packet Delivery Service on which the MBMS client failed.
- `PacketErrorCode errorCode` – identifies the error code for the reason causing the `startPacketService()` or the `stopPacketService()` request for the serviceId to fail. The available error codes are:
  - `Packet_INVALID_SERVICE` – signals that serviceID defined on the `startPacketService()` or the `stopPacketService()` request is not currently defined or it is not associated with the service classes with the MAA is registered.
  - `Packet_UNKNOWN_ERROR` – signals an error condition not explicitly identified.
- `string errorMsg` – may provide additional textual description of the error condition.

6.4.3.12.3 Pre-Conditions

The MBMS client has received a the `startPacketService()` or a `stopPacketService()` request.

6.4.3.12.4 Expected MBMS Client Actions

The MBMS client will send a failure indication via the `packetServiceError()` to signal the error code for the result of processing the MAA. For more details refer to clauses 6.4.2.4 and 6.4.2.5.

6.4.3.12.5 Usage of Method for MAA

If the MAA receives this notification, it should revalidate the capture call. The MAA should also update the service list by issuing a `getPacketServices()` as defined in clause 6.4.3.4.

6.4.3.12.6 Post-Conditions

No state change is applied.

6.4.3.13 Checking the version for MBMS Packet Delivery Service interface

6.4.3.13.1 Overview

This clause defines the `getVersion()` request function.

6.4.3.13.2 Parameters

The parameters for the `getVersion()`API call are:

- `string version` – identifies the version of the MBMS clients API implementation.

6.4.3.13.3 Pre-Conditions

The MBMS client may be in any state.

6.4.3.13.4 Usage of Method for MAA

In order for the MAA to know the version of the MBMS Packet Delivery Service interface, the `getVersion()` API may be used. If the version number is not supported by the MAA, it should deregister and not use the API.

6.4.3.13.5 MBMS Client Actions

The `getVersion()` API returns the version of the implemented APIs of the MBMS client.

6.4.3.13.6 Post-Conditions

No state changes apply.

7 MBMS Client to Application Interfaces for Data

7.1 General

The MBMS client when providing a DASH Streaming Application User Services should implement the generic HTTP interface as defined in clause 7.3.

In this case, the MBMS client shall implement at least on of the following:

- the functions of a DASH server as defined in clause 7.4.2, or
- the functions of a DASH-aware Network Element (DANE) for SAND4M mode as defined inclause 7.4.3. .

The MBMS client may act as a proxy and may proxy all HTTP requests from the DASH client or may only proxy selected requests, such as MPD updates. The details of operation are implementation specific.

The requirements for the DASH client are documented in clause 7.4.4.

For additional usage guidelines refer to TS 26.346 [5], Annex K and in this specification refer to Annex E.

7.2 File Copy Interface

The MBMS client may copy files delivered by MBMS User Services to a local file storage that is controlled by the MBMS aware application. The application is responsible for the management of the storage.

7.3 HTTP Interface

The MBMS client may provide an HTTP server such that the MAA can access the files delivered over the MBMS User services by using regular HTTP Methods. The MBMS client may act as an HTTP cache for the resources delivered through the MBMS system.

The MBMS client offering delivered files and DASH Segments through HTTP Interface should comply with a *server* as specified in RFC 2616 [15]. MBMS Clients providing resources through an HTTP interface should implement relevant HTTP server functionalities to support HTTP GET methods as required by the APIs.

MAAs communicating with the MBMS client over HTTP should comply with a *client* as specified in RFC 2616 [15]. MAAs should use the HTTP GET method or the HTTP partial GET method, as specified in RFC 2616 [15] to access files offered at HTTP-URLs.

MAAs communicating with the MBMS client over HTTP should support partial-file-accept requests and partial file responses as defined in TS 26.346 [5], clause 7.9.2.1.

Without excluding other response options, as a response to a partial-file-accept request using a regular HTTP GET request an MAA may typically receive one of the following responses:

1) 200 OK with Content-Type set to the Media Type of the requested object

2) 200 OK with the Content-Type set to `application/3gpp-partial` and the message format according to the definition in clause 7.9.2.2 of TS 26.346 [5].

3) `416 Requested Range Not Satisfiable` with the additional information according to the definition in clause 7.9.2.2 of TS 26.346 [5].

4) `404 Not Found.`

Case 1 is the regular response.

Guidelines for handling request responses according to case 4 from above are provided in clause A.7 of TS 26.247 [6].

Guidelines for handling request responses 2 and 3 from above are provided in clause A.9 of TS 26.247 [6].

7.4 DASH-Specific Interfaces

7.4.1 General

The MBMS client when providing a DASH Streaming Application User Services should implement the generic HTTP interface as defined in clause 7.3.

In this case, the MBMS client shall implement at least one of the following:

- the functions of a DASH server as defined in clause 7.4.2, or
- the functions of a DASH-aware Network Element (DANE) for SAND4M mode as defined inclause 7.4.3. .

The MBMS client may act as a proxy and may proxy all HTTP requests from the DASH client or may only proxy selected requests, such as MPD updates. The details of operation are implementation specific.

The requirements for the DASH client are documented in clause 7.4.4.

For additional usage guidelines refer to TS 26.346 [5], Annex K and in this specification refer to Annex E.

7.4.2 MBMS Client as DASH Server

7.4.2.1 General

If the MBMS client supports the DASH Server functionalities, the MBMS client shall support sufficient functionalities to provide a valid DASH Media Presentation as defined in TS26.247[7].

For this purpose, the MBMS client may rewrite the MPD to provide a valid service based on the data received in the DASH-over-MBMS service. Typical operations that the MBMS client may implement:

- Ensure that the URLs of the DASH Resources in the MPD resolve to the location at which the MBMS client provides the resources delivered through the MBMS User Service,
- Ensure that the timeshift buffer expressed in the MPD does not refer to non-available resources,
- Ensure that the availability times of the DASH resources on the MBMS clients DASH Server are correctly documented in the MPD,
- Ensure that MPD Updates are done sufficiently often, for example by setting the `@minimumUpdatePeriod` to 0 in order for the DASH client to revalidate the MPD prior to every Segment request.

Some specific functions are provided in the remainder of clause 7.4.2.

7.4.2.2 Time Synchronization

The MBMS client, if providing the content as a DASH server, should offer the Time Synchronization as defined in clause 11.5.2 of TS 26.247 [7].

7.4.2.3 Robustness

The MBMS client if providing the content as a DASH server may implement the Robustness Tools on the server as defined in clause 11.6 of TS 26.247 [7].

7.4.3 MBMS Client as DASH-Aware Network Element (DANE)

If the MBMS client acts as a DANE, it shall implement the DANE SAND4M functionalities as required in clause 13.10.2 of TS 26.247 [7].

If the MBMS client identifies that the DASH client ignores the SAND messages, it should use other means to provide the functionality for redirecting the client, e.g. the functions defined in clause 7.4.2.

7.4.4 DASH Client of MBMS-Aware Application

The DASH client of the MBMS aware application shall support general DASH client functionalities as required in TS 26.247 [7].

The DASH client of the MBMS aware application should support the Time Synchronization as defined in clause 11.5.3 of TS 26.247 [7].

The DASH client of the MBMS aware application should support the Robustness Tools on the client as defined in clause 11.5.3 of TS 26.247 [7].

The DASH client of the MBMS-aware application should support the the SAND4M functionalities in clause 13.10.3 of TS 26.247 [7].

7.5 RTP Streaming Delivery Method Interface

The MBMS Client should provide an interface such that the data delivered using the MBMS Streaming delivery method is provided as a packet stream that complies with a media format that can be decoded by the media receiver that is part of the MBMS client by offering a conforming SDP in the `sdpURI`. As a recommendation:

- The MBMS Client implements the functions of the "Hypothetical FEC Decoder" as defined in clause 8.2.2.11 of TS 26.346 [5],
- The MBMS Client provides the MAA with:
  - A SDP that describes the packet stream.
  - The network interface from which the data stream can be received. For that purpose, the MBMS Client may forward the packets locally, e.g. through a virtual network interface, or through the network to the client. In such an example, the MBMS client is expected to modify the SDP provided over in the User Service Description accordingly.

7.6 Packet Data Interface

The MBMS Client should provide an interface such that the data delivered using the MBMS Transparent delivery method can provide a packet stream that complies with the SDP provided in the value of the `sdpURI`.

The MBMS Client provides the application with:

- A SDP that describes the data stream.
- The network interface from which the data stream can be received. For that purpose, the MBMS Client may forward the packets locally, e.g. through a virtual network interface, or through the network to the client. In such example, the MBMS client is expected to modify the SDP accordingly to provide a session that conforms to the offered SDP.

8 MBMS URLs: Definition and URL Handling

8.1 General

This clause defines different MBMS URLs and the associated URL handling.

The single resource MBMS URL form "mbms://" is defined in clause 8.2.

NOTE: The specification does not define a URL for all Application User Services defined in clause 4.3, but only a subset.

8.2 Single Resource MBMS URL handling

8.2.1 Introduction

An MBMS URL identifies a resource that is made available over MBMS Download Delivery Service using the File Download Application User Service as defined in clause 4.2.3. It is therefore similar in meaning to other URL forms that deliver 'file' resources, such as "http:", "ftp:" and "file:".

MBMS URLs are processed by a hypothetical MBMS URL Handler as shown in Figure 8.2.1-1.

The position of the MBMS URL Handler, and its interfaces are illustrated by the logical model shown in Figure 8.2.1-1. In that figure, URL usage is shown as an alternative to an application written to use MBMS services using the MBMS-API (left side of the diagram, described elsewhere in the present document).

The MBMS URL handler is positioned within the logical model of a system that has library support for fetching resources (files) referred to by URLs. In the model an (unchanged) existing, or new, application supports URLs that address 'file' resources. That application uses a generic URL resolution library ('Generic URL Resolution' in the diagram) to return the identified resource when the application needs the resource identified by a URL.

The MBMS URL handler may use MBMS-API functions to communicate with the MBMS client.

The logical model is that the generic handler supports returning the resource given a URL of any type. That generic handler is supported by a set of protocol-specific handlers; by inspecting the scheme part of the URL (e.g. "http:") the generic handler in turn requests the resource of the appropriate protocol-specific handler. The request and response interfaces to the generic handler, and the protocol-specific modules are defined by the library (e.g. they may be object-oriented, function-based, or message-based). The present document defines the behaviour of the MBMS URL handler; however, its request interface is defined by the library and environment which it fits into. The MBMS protocol-specific handler decomposes the URL form, and, using the MBMS-APIs, initiates the acquisition of the MBMS User service by the MBMS client that permits access to the identified resource, and acquisition of the indicated resource from that service, and returns that resource. The behaviour of the MBMS URL handler is specified in this clause.

Note that this logical model may be optimized and collapsed as desired in real deployments (e.g. many web browsers do not use a general URL library but instead support key protocols with built-in code, and the MBMS function may also act as an MBMS URL handler).

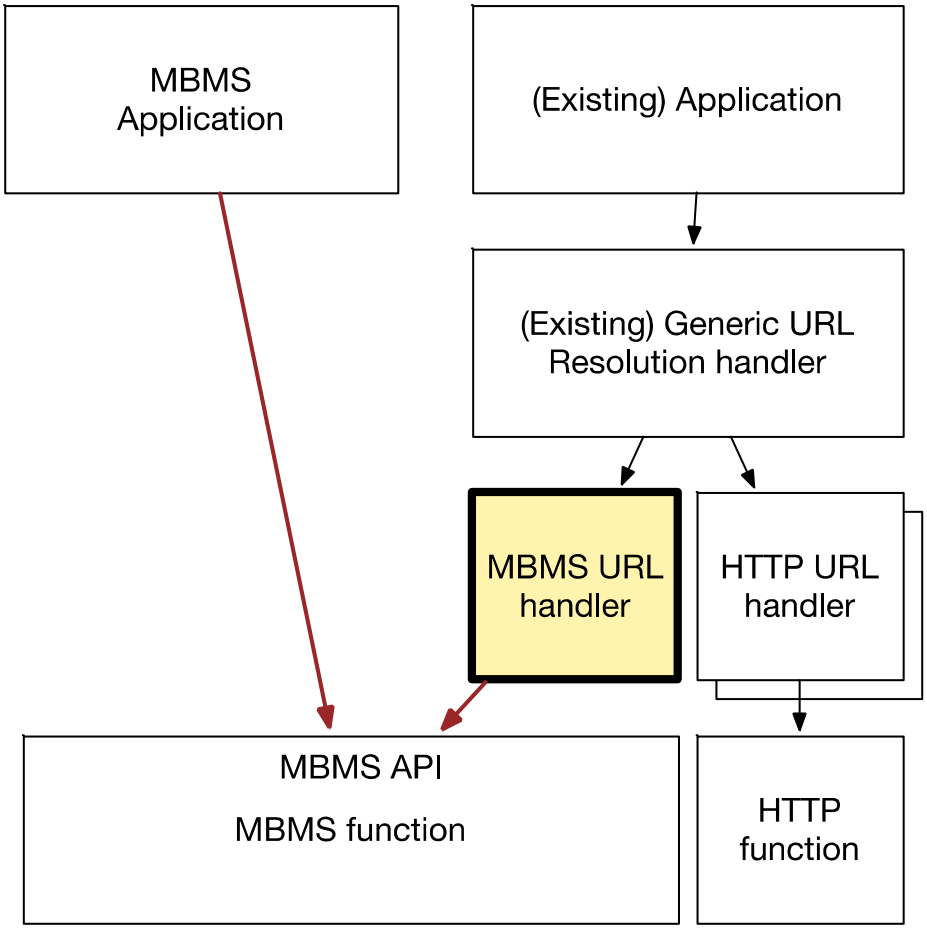

Figure 8.2.1-1: MBMS URL Handler

8.2.2 URL structure, definition and behaviour

The Single Resource MBMS URL is composed of a prefix, mid-part, and suffix.

The prefix is a URL that is the `serviceId` of the service on which the resource is available. The suffix contains a URI that is the identifier of the desired resource. The mid-part currently has no defined content.

When a content provider wishes to use MBMS URL forms to identify resources made available by them over MBMS, they shall use the prefix part of an MBMS URL as `serviceId` for the service. The creator of the `serviceId` URI shall be authorized by the 'authority' (see below) identified in that `serviceId`.

The prefix starts with the scheme-name `"mbms://"` followed by a double-slash `"//"`, followed by an authority and an optional path, as defined for `"http:"` by RFC 7230 [14] with the restriction that the prefix shall not contain the character "&".

NOTE: The "//" means that the MBMS scheme is hierarchical and that relative URLs are permitted, and that they are effectively composed against the suffix part. Relative URL composition is not in the scope of the present document; a relative URL is, of course, composed against its base, and if the final result is an MBMS URL as defined here, then that URL is presented to the handler.

The mid-part consists of zero or more `&name=value` pairs. There are no currently defined mid-part pairs; they shall not be present in URLs and shall be ignored if present.

The suffix is optional and consists of the string `"&label="` followed by the URI which contains identifier of the desired resource (and optionally, as for all URIs, a query or fragment part). The suffix is terminated by the end of the URL.

(Note that the use of `"&"` used here is ***not*** part of a query or fragment, as there is no preceding `"?"` or `"#"`. `"&"` is in the main body of the URL where it is a legal character.)

The formal ABNF, following the syntax of RFC 7230 (HTTP URLs) [14] and RFC 3986 (URLs) [11]:

```
mbms-URI     = "mbms:" "//" authority path-abempty
   *( "&" mid-label "=" mid-value )
   [ "&label=" resourceURI ]
authority    = <authority, see [RFC3986], Section 3.2>
path-abempty = <path-abempty, see [RFC3986], Section 3.3>
mid-label    = ALPHA *( ALPHA / DIGIT )
mid-value    = 1*uchar
        =  unreserved / pct-encoded / ";" / "?" / ":" /
   "@" / "=" / "+" / "$" / "," / "/"
unreserved   = <unreserved, see [RFC3986], Section 2.3>
pct-encoded  = <pct-encoded, see [RFC3986], Section 2.1>
resourceURI  = <URI, see [RFC3986], Section 3>
```

The handler performs decomposition of the URI into the prefix, mid-part, and suffix.

If the URI passed to the handler lacks the path and suffix part (there is no `"&label="`), then the handler retrieves the DNS URI record as defined in section 8.2.2A.

If this operation succeeds in building a new URI that has a suffix, the handler returns a Redirect to this new URI; otherwise it returns an error.

Otherwise, the `serviceId` is set to the prefix (the substring of the URI before the first `"&"`) and the resourceLabel is set to portion of the resourceURI preceding the fragment part, if any. The MBMS API is called to return the resource identified by resourceLabel from the service identified by `serviceId`.

The MBMS-API and the MBMS Client are responsible for:

1) finding the service with the given `serviceId` among the available services; the serviceClass is empty. This is done by

   a) registering;

   b) asking for the available services;

   c) issuing a start capture.

2) checking the availability time interval of the resource;

   a) provide the schedule as defined in.

3) finding and returning to the MBMS URL Handler the resource on the given service;

   a) notify the MBMS URL handler by the notification if successful or notification if an error occurs.

The MBMS URL handler is responsible for:

1) keeping the service open for some "keep-alive" time in the expectation of further calls for resources from the same service, by staying registered;

2) caching resources received on the service;

3) sharing that cache with other protocol handlers;

4) returning the resource, or returning errors or redirects, translating the return value(s) from the MBMS APIs as appropriate.

8.2.2A DNS URL RR Resolution

An MBMS URL that lacks the label suffix shall be resolved using an URI RR DNS query as defined in RFC 7553 [15]. The service name shall be “mbms” and the protocol shall be UDP. The DNS query shall be made to the FQDN of the MBMS URL using a query type “URI”. The query name shall be the FQDN prepended with “*_mbms._udp*”.

If found, the DNS response shall contain the new URL that shall match the service ID of the service that carries the resource with the requested URL. Otherwise, the DNS response shall have no DNS URI record. For mbms URLs that do not have a path part (other than "/"), the requested resource's URL will correspond to the label part of the URL in the DNS response.

8.2.3 Examples

The following is an example of an MBMS URL that contains a label:

`mbms://service1000.mbms.operator.com&label=http://www.example.com/videos/sample.mp4`

- the `serviceId` is `mbms://service1000.mbms.operator.com;`
- the label of the desired resource is `http://www.example.com/videos/sample.mp4.`

An example of the same resource that does not have a label and a path part is as follows:

`mbms://www.example.com/`

This MBMS URL is then resolved by using the DNS URI RR query.

In this example, "example.com" has a business arrangement with "operator.com" for hosting the service. An example DNS URI RR record may look as follows:

```
$ORIGIN www.example.com.

_mbms._udp   IN      URI     3600    1       "mbms://service1000.mbms.operator.com/videos/service.mpd&label=http://www.example.com/videos/sample.mp4"
```

After resolving the URL using a DNS URI RR query, the client is re-directed to the URL returned:`mbms://service100.mbms.operator.com&label=http://www.example.com/videos/sample.mp4`

It can then establish both the service name, and the label of the initial resource.

Annex A (informative): Documentation Guidelines for APIs

A.1 Introduction Motivation

As the present document is primarily targeted at developers that use the MBMS function a few principle agreements were made.

1) The specification should set good practices for future API definition in 3GPP
2) The specification requires clear definitions of interface, function/element, messages, API, resource, etc.
3) The interfaces should be documented in the following manner:
   a) Consistent graphical presentation
   b) IDL for interface and message description
   c) Doxygen for semantical description
   d) Example Use Cases and Message Flows
4) The semantical description and messages flows include messages sent and received on each function as well as pre/post/error conditions.
5) As IDL enables easy conversion to different languages, no program-language specific APIs (e.g. JSON, Java, or C) are necessary, but may in general not be prohibited. If IDL can provide a non-ambiguous translation to JSON, Java and C, then the documentation may be informative, otherwise it may be added to the normative specification as a reference able option.

Based on these considerations, some suggested documentation details are provided

A.2 Documentation Details

A.2.0 General

A graphical presentation of the interface is encouraged. Figure A.1 shows some examples using an informal notation. The existence of an interface can be implied even without using an explicit symbol for it. If a relationship symbol joins an element/function symbol and the relationship type involves an interaction—as opposed to, say, "is a subclass of," that implies that the interaction takes place through the element's interface.

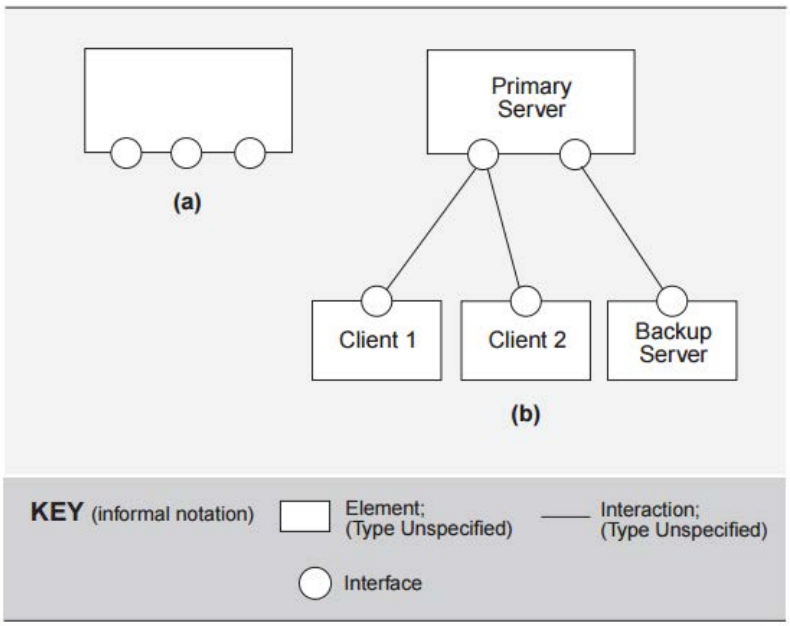

Figure A.1: Example Figure for Interface documentation

A.2.1 IDL for Interface Specification

Interface Definition Language (IDL) [9] is a standard language for defining function and method interfaces. It supports common primitive data types. Some of the base data types supported by IDL are `int`, `Boolean`, `byte`, `char`, `double`, `float`, `long`, `short`, and `void` *. IDL also supports signed and unsigned qualifiers, enumerations and more.

IDL's data types and definitions are both language-neutral and platform-neutral. This enables the definition of a common interface that may be implemented by different modules or components irrespective of the language and platform. An IDL interface provides a description of the functionality that will be provided by an object. It provides all of the information needed to develop clients that use the interface.

While IDL is not a programming language, there are tools that map IDL to just about every major programming language.

Below is a sample IDL definition of "order"

```
interface order {
   float calculate_tax ([in] float taxable_amount);
   float calculate_total([in] item_list items);
   bool place_order([in,out] item_list items);
}
```

A.2.2 IDL as Data Format

IDL is also proposed as the data format for message exchange.

A.2.3 Doxygen for API Semantics

Doxygen is a cross-platform documentation system for C++, Java, C, and IDL, etc. It provides a mechanism to document code syntax and semantics. It allows tagging comments in code that will be used to generate nicely formatted output. Available tags enables capturing methods, parameters, pre and post conditions for calling methods as well as highlighting error condition and exceptions.

Doxygen can be used to generate on-line class browser (in HTML) and/or an off-line reference manual from a set of source files. Below is a sample of Doxygen comments for a function definition.

```
/** @name binary_search
```

```
 *   @brief Search a  array for a value using binary search.
 *   @param[in] a Array of floats.
 *   @param[in] n Number of elements of @a a to search.
 *   @param[in] v Value to search for.
 *   @return An index into array @a a or -1.
 *
 *   @pre 0 <= n <= Size of the array @a a.
 *   @pre For all i,j with 0 <= i < j < @a n, @a a[i] <= @a a[j].
 *   @post (0 <= result < @a n and @a a[result] == @a v)
 *     or (result == -1 and there is no i, 0 <= i < @a n, s.t. @a a[i] == @a v).
 */
int binary_search(const float* a, int n, float v)
```

A.2.4 Use Cases and Message Flows

Message flows (sequence diagrams) are an interaction diagram that shows the objects/modules participating in a particular interaction and the messages they exchange arranged in a time sequence.

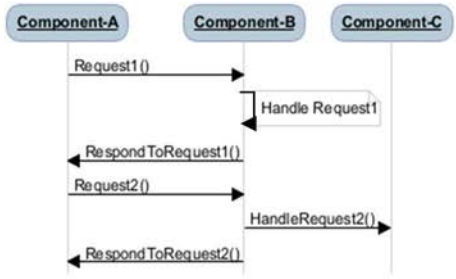

Figure A.2: Example Figure for Sequence Diagram

Annex B (informative): Interface Definition Language for MBMS-APIs

B.1 General

This Annex collects the Interface Definition Language (IDL) description for MBMS-APIs.

B.2 IDL for File Delivery Application Service API

```
#include "EmbmsCommonTypes.idl"

module FileDeliveryService
{
    //Forward Declaration
    interface  ILTEFileDeliveryServiceCallback;

    /**
    * @name DownloadState
    * @brief List of the file download state
    */
    enum DownloadState
    {
        FD_IN_PROGRESS  /**< File download is in progress              */
    };

    /**
    * @name FdErrorCode
    * @brief List of the errors for File Delivery service
    */
    enum FdErrorCode
    {
        FD_INVALID_SERVICE,  /**< Invalid service ID                     */
        FD_DUPLICATE_FILE_URI, /**< There is another pending capture request for the specified file URI. */
        FD_AMBIGUOUS_FILE_URI,  /**< The specified file URI cannot identify a pending capture request. */
          FD_STOP_FILE_URI_NOT_FOUND,  /**< The file URI specified on a stopFdCapture does not match an
       outstanding startFdCapture() request. */
        FD_UNKNOWN_ERROR     /**< Unknown error                          */
    };

    /**
    * @name cacheControl
    * @brief List of the errors for File Delivery service
    */
    enum cacheControlMode
    {
        FD_NO_CACHE,        /**< The application uses Cache directives to manage how long to retain files.
                                 When FD_NO_CACHE is selected, the file (or set of files)
                                 won't be cached, which can be useful when the file is expected to be
                                 highly dynamic (changes to the file occur quite often) or if the file
                                 will be used only once by the receiver application. */
        FD_MAX_STALE,       /**< The application uses Cache directives to manage how long to retain files.
                                  When FD_MAX_STALE is selected, the file (or set of files)
                                  won't be cached, which can be useful when the file is expected to be
                                  highly dynamic (changes to the file occur quite often) or if the file
                                  will be used only once by the receiver application. */
        FD_EXPIRES             /**< The application uses Cache directives to manage how long to retain files.
                                  When FD_EXPIRES is selected, indicates the file has expected expiry time.
                                  In that case cacheControlExpires value is the expiry time*/
    };
    /**
    * @name RegisterFdResponseNotification
    * @brief Fd app registration information
    */
    struct RegisterFdResponseNotification
    {
        EmbmsCommonTypes::RegResponseCode value;    /**< Result of registration  value as defined in
RegResponseCode     */
        string message;                             /**< Message described the result
*/
```

```
        unsigned long   acceptedFdRegistrationValidityDuration; /**< Accepted registeration validity duration
*/
    };

    /**
    * @name FileInfo
    * @brief Downloaded file information
    */
    struct FileInfo
    {
        string fileUri;         /**< File URI
*/
        string fileLocation;    /**< The physical location of the file or HTTP URL where the file can be
accessed    */
        string contentType;     /**< MIME type as described in FDT of the file
*/

        unsigned long availabilityDeadline;  /**< The maximum time that embms client guarantees to keep the
file in its storage   */

    };

    /**
    * @name RegisterFdAppData
    * @brief File delivery app registration information
    */
    struct RegisterFdAppData
    {
        string appId;                       /**< The application ID used during the registration
*/
        any    platformSpecificAppContext;  /**< The platformSpecificAppContext provides a
                                             platform-specific app context
                                             object to enable the API implementation to get extra
information
                                             about the application. */


        sequence<string> serviceClassList;    /**< List of service classes                 */

        StorageLocation locationPath;        /**< Local storage location on the device where collected files
are copied */

        unsigned long registrationValidityDuration;  /**<  The period of time in seconds that the eMBMS
client honors
                                                        the app registration and file capture requests
                                                        after the app deregisters and exits.
                                                        This enables the app to let the eMBMS client capture
                                                        files in the background when the application is not
currently registered.
                                                        Default value of this option is 0 which means
middleware clears
                                                        any outstanding startFdCapture requests.* /
    };


    /**
    * @name StartFdCaptureData
    * @brief File delivery start capture information. It is used in StartFdCapture API
    */
    struct StartFdCaptureData
    {
        string serviceId;        /**< File delivery service ID from FdServiceInfo              */
        string fileUri;          /**< File URI for the file(s) to be captured.
                                 If empty, this implies capture all files. If an absolute URL,
                                 this implies only the capture of that particular file.
                                 If a Base URL, this implies the capture of all files that have that Base
URL. */
        boolean disableFileCopy;      /**< Disables copying of files to register locationPath */
        boolean captureOnce;    /**< Capture the file only once and the bearer would be deactivated after
file gets downloaded*/
    };


    /**
    * @name StopFdCaptureData
    * @brief File delivery stop capture information. It is used in StopFdCapture API
```

```
    */
    struct StopFdCaptureData
    {
        string serviceId;   /**< File delivery service ID from FdServiceInfo                */
        string fileUri;     /**< File URI. If empty, then it stops capture on all files.
                                 The path of the URI should contain the complete folder or file name.     */
    };


    /**
    * @name FileList
    * @brief List of file URIs
    */
    struct FileList
    {
        sequence<string> fileUriList;  /**< List of file URIs                       */
    };


    /**
    * @name ServiceNameLang
    * @brief Name and language information
    */
    struct ServiceNameLang
    {
        string name;        /**< Name                 */
        string lang;        /**< Language             */
    };

    /**
    * @name FdServiceInfo
    * @brief File delivery service information
    */
    struct FdServiceInfo
    {
        sequence<ServiceNameLang> serviceNameList;      /**< List of Service name and language          */
        string serviceClass;                            /**< Service class                              */
        string serviceId;                               /**< Service ID                                 */
        string serviceLanguage;                         /**< Service language                           */
        EmbmsCommonTypes::ServiceAvailabilityType serviceBroadcastAvailability;    /**< Service broadcast
availability     */
        sequence<string> fileUriList;                   /**< List of file URIs                          */
        EmbmsCommonTypes::Date activeDownloadPeriodStartTime;    /**< The current/next active file download
service start time, when files start being broadcast over the air       */
        EmbmsCommonTypes::Date activeDownloadPeriodEndTime;           /**< The current/next active file
download service end time, when files stop being broadcast over the air         */

    };

    /**
    * @name FdServices
    * @brief List of FD service info objects
    */
    typedef  sequence<FdServiceInfo> FdServices;

    /**
    * @name FdServiceClassList
    * @brief ServiceClass information that the app is interested in. It is for the SetFdServiceClassFilter
API.
    */
    typedef sequence<string> FdServiceClassList;

    /**
    * @name ActiveFdService
    * @brief Information about active file capture
    */
    struct ActiveFdService
    {
        string  serviceId;    /**< File delivery service ID from FdServiceInfo        */
        sequence< string >  fileUri;      /**< File URI list                                      */
    };

    /**
    * @name ActiveFdServiceList
    * @brief List of File delivery service ID from FdServiceInfo
    * @see getFdActiveServices()
```

```
    */
    typedef sequence< ActiveFdService >  ActiveFdServiceList;


    /**
    * @name StorageLocation
    * @brief  Local storage location on the device where collected files are copied.
    *       It is used in the SetStorageLocation and registerFdApp API.
    */
    typedef string StorageLocation;



    /**
    * @name FileAvailableNotification
    * @brief Information about the downloaded file.
    */
    struct FileAvailableNotification
    {
        string  serviceId;    /**< File delivery service ID from FdServiceInfo        */
        FileInfo downloadedFileInfo;   /**< Downloaded file information                        */
    };


    /**
    * @name FdServiceErrorNotification
    * @brief File delivery service error information. It is used by the FdServiceErrorNotification API.
    */
    struct FdServiceErrorNotification
    {
        string serviceId;    /**< File delivery service ID from FdServiceInfo       */
        string fileUri;      /**< File URI                                          */
        FdErrorCode errorCode;     /**< File delivery service error ID                    */
        string errorMsg;      /**< error message                                     */
    };

    /**
    * @name FileDownloadFailureNotification
    * @brief File download failure information.
    * @see FileDownloadFailureNotification()
    */
    struct FileDownloadFailureNotification
    {
        string serviceId;   /**< File delivery service ID from FdServiceInfo        */
        string fileUri;     /**< File URI                                           */
    };

    /**
    * @name StorageError
    * @brief Insufficient storage notification information
    * @see StorageError()
    */
    struct StorageErrorNotification
    {
        string serviceId;       /**< File delivery service ID from FdServiceInfo        */
        string fileUri;         /**< File URI                                           */
        StorageLocation storagePath;     /**< Storage path that does not have sufficient storage to complete
the file download  */
        unsigned long  storageNeeded; /**< Storage needed to complete the file download
*/
    };

    /**
    * @name InaccessibleLocationNotification
    * @brief Inaccessible storage notification information
    * @see InaccessibleLocation()
    */
    struct InaccessibleLocationNotification
    {
        string serviceId;    /**< File delivery service ID from FdServiceInfo         */
        string message;      /**< Message with additional information                */
        StorageLocation locationPath; /**< The path that is not accessible                    */
    };

    /**
    * @name FdDownloadStateInfo
```

```
    * @brief Information returned by getFdDownloadStateList().
    * @see getFdDownloadStateList()
    */
    struct FdDownloadStateInfo
    {
        string fileUri;       /**< File URI    */
        DownloadState state;  /**< State of files from DownloadState.        */
    };

    /**
    * @name FileDownloadStateInfoList
    * @brief List of FdDownloadStateInfo
    * @see getFdDownloadStateList()
    */
    typedef sequence<FdDownloadStateInfo> FileDownloadStateInfoList;


    /**
    * @name FileDownloadStateUpdateNotification
    * @brief File download state update notification information
    * @see fileDownloadStateUpdate()
    */
    struct FileDownloadStateUpdateNotification
    {
        string serviceId;  /**< File delivery service ID from FdServiceInfo         */

    };

    /**
    * @name GetFdDownloadStateListData
    * @brief Information needed to call getFdDownloadStateList(). The returned list of
getFdDownloadStateList() is filtered based on the options set in GetFdDownloadStateList.
    * @see getFdDownloadStateList()
    */
    struct GetFdDownloadStateListData
    {
        string serviceId;  /**< Active file delivery service ID from FdServiceInfo.         */
    };


    /**
    * @name AvailableFileList
    * @brief List of FileInfo
    * @see getFdAvailableFileList()
    */
    typedef sequence < FileInfo > AvailableFileList;


    /**
    * @name FileListAvailableNotification
    * @brief File List Available notification information
    * @see fileListAvailable()
    */
    struct FileListAvailableNotification
    {
        string serviceId;  /**<File delivery service ID from FdServiceInfo.         */
    };


    interface ILTEFileDeliveryService
    {

        /**
        @name getVersion
        @brief Retrieves the version of the current File delivery service interface implementation
        @return Interface Version
        **/
        string getVersion();

        /**
        @name registerFdApp
        @brief Application registers a callback listener with the EMBMS client
        @param[in] regInfo Information required for application registration
        @param[in] cb Callback listener
        @see RegisterFdAppData
        @see registerFdResponse()
```

```
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode registerFdApp(in RegisterFdAppData regInfo, in
ILTEFileDeliveryServiceCallback  callBack);

        /**
        @name deregisterFdApp
        @brief Application deregisters with the EMBMS client
        @pre Application calls registerFdApp
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode deregisterFdApp();



        /**
        @name startFdCapture
        @brief Start download of files over file delivery service over broadcast
        @param StartFdCapture Struct includes parameters for StartFdCapture request
        @pre Application is registered for File Delivery service
        @see fileAvailable()
        @see StartFdCaptureData
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode startFdCapture(in StartFdCaptureData info);


        /**
        @name stopFdCapture
        @brief Stop download of files for the file Delivery service over broadcast
        @param stopFdCapture Struct includes parameters for stopFdCapture
        @pre Application is registered for File Delivery service
        @see StopFdCaptureData
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode stopFdCapture(in StopFdCaptureData info);

        /**
        @name getFdActiveServices
        @brief Get list of currently active services
        @param[out] ActiveFdServiceList The list of services the app has
        @pre Application is registered for File delivery service
        @see ActiveFdServiceList
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode getFdActiveServices(out ActiveFdServiceList services);

        /**
        @name getFdAvailableFileList
        @brief Retrieves the list of files previously captured for the
        application.
        @param[in] File delivery service ID from FdServiceInfo
        @param[out] FileList List of files previously captured and filtered based on serviceId
        @pre Application is registered for File delivery service and received fileListAvailable()
notification
        @see fileListAvailable()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode getFdAvailableFileList(in string serviceId, out AvailableFileList
files);

        /**
        @name getFdServices
        @brief Retrieves the list of File Delivery services defined in the USD.
        List of services is filtered by the service class filter,
        if a filter has been set by the application
        @param[out]  FDServices List of filtered File delivery services
        @pre Application is registered for File delivery service and received fdServiceListUpdate()
notification
        @see fdServiceListUpdate()
        @see FdServices
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode getFdServices(out FdServices services);

        /**
```

```
        @name getFdDownloadStateList
        @brief Retrieves the state of files pending download
        @param GetFileDownloadState Includes parameters for getFileDownloadState
        @pre Application is registered for File Delivery service and received fileDownloadStateUpdate()
notification
        @see fileDownloadStateUpdate()
        @see GetFdDownloadStateListData
        @see FileDownloadStateInfoList
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode getFdDownloadStateList(in GetFdDownloadStateListData info, out
FileDownloadStateInfoList fdStateList);

        /**
        @name setFdServiceClassFilter
        @brief Application sets a filter on file delivery services in which it is interested
        @param[in] serviceClassInfo List of service class filters requested by the application
        @pre Application is registered successfully with file delivery service
        @see SetFdServiceClassFilterData
        @see fdServiceListUpdate()
        @see getFdServices()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode setFdServiceClassFilter(in FdServiceClassList serviceClassInfo);


        /**
        @name setFdStorageLocation
        @brief Sets the storage location to store the application downloaded files
        @param[in] StorageLocation Includes parameters for setStorageLocation request
        @pre Application is registered for File Delivery service
        @see StorageLocation
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode setFdStorageLocation(in StorageLocation locationPath);

    };

    interface ILTEFileDeliveryServiceCallback
    {
        /**
        @name registerFdResponse
        @brief The response to the application streaming service register API.
        @param  Notification Parameters for register File delivery response
        @pre Application called registerFdApp
        @see RegisterFdResponseNotification
        @see registerFdApp()
        **/
        void registerFdResponse(in RegisterFdResponseNotification info);

        /**
        @name fileAvailable
        @brief Notification to application when a new file is downloaded per
        application capture request
        @param FileAvailableNotification Includes parameters for the downloaded file
        @pre Application is registered for File Delivery service and application called startFdCapture()
        @see FileAvailableNotification
        **/
        void fileAvailable(in FileAvailableNotification notification);

        /**
        @name fdServiceListUpdate
        @brief Notification to application on an update of the available for file delivery services.
        Update may be due to the received USD or the network configuration
        @pre Application is registered for file delivery service
        @post Call getFdServices()
        **/
        void fdServiceListUpdate();


        /**
        @name fdServiceError
        @brief Notification to application when there is an error with broadcast download of service
        @param Notification Parameters for service error notification
        @pre Application is registered for streaming service and called startFdServiceCapture
        @see FdServiceErrorNotification
```

```
        **/
        void fdServiceError(in FdServiceErrorNotification notification);

        /**
        @name fileDownloadFailure
        @brief Notification to application that download of a requested file
        failed
        @param  FileDownloadFailureNotification Includes information about the failed file download
        @pre Application is registered for File Delivery service and application called startFdCapture()
        @see FileDownloadFailureNotification
        **/
        void fileDownloadFailure(in FileDownloadFailureNotification notification);

        /**
        @name storageError
        @brief Notification to application that the storage location set by the
        application does not have enough storage for the file download
        @param StorageError Includes parameters to specify the file and
        storage requirement
        @pre Application is registered for file delivery service and application called startFdCapture()
        @see StorageError
        **/
        void storageError(in StorageErrorNotification info);

        /**
        @name inaccessibleLocation
        @brief Notification to application that the storage location set by the
        application is not accessible by the eMBMS Client
        @param InaccessibleLocation Includes the inaccessible storage path
        @pre Application is registered for File delivery service
        @see InaccessibleLocation
        Application calls setStorageLocation
        **/
        void inaccessibleLocation(in InaccessibleLocationNotification info);

        /**
        @name fileDownloadStateUpdate
        @brief Notify application of a change in the state of pending file
        downloads
        @param FileDownloadStateUpdate Includes parameters for fileDownloadStateUpdate()
        @pre Application is registered for File delivery service
        @post call getFdDownloadStateList()
        @see FileDownloadStateUpdate
        **/
        void fileDownloadStateUpdate(in FileDownloadStateUpdateNotification info);

        /**
        @name fileListAvailable
        @brief Notify application when the list of downloaded files is available to retrieve
        @param[in] FileListAvailable Includes parameters for fileListAvailable
        @pre Application is registered for File Delivery service
        @post call getFdAvailableFileList()
        **/
        void fileListAvailable(in FileListAvailableNotification info);
    };

};


module EmbmsCommonTypes
{
      //Common types
    typedef unsigned long long Date;

    /**
    * @name ResultCode
    * @brief The return value of the API
    */
    enum ResultCode
    {
        SUCCESS,                    /**< Success                          */
        REGISTRATION_IN_PROGRESS,   /**< Failed due to registration in progress       */
        NO_VALID_REGISTRATION,      /**< Failed due to no valid registration      */
        MISSING_PARAMETER,          /**< A mandatory parameter is missing             */
UNKNOWN_ERROR               /**< Failed with unknown error      */
    };
```

```
    /**
    * @name ServiceAvailabilityType
    * @brief Indicates service availability state
    */
    enum ServiceAvailabilityType
    {
        BROADCAST_AVAILABLE,   /**< Service is available via broadcast            */
        BROADCAST_UNAVAILABLE, /**< Service is unavailable via broadcast          */     SERVICE_UNAVAILABLE
/**< Service is unavailable                        */
    };

    /**
    * @name RegResponseCode
    * @brief Indicates app registration response
    */
    enum RegResponseCode
    {
        REGISTER_SUCCESS,                     /**< Registration was successful
*/
        FAILED_LTE_EMBMS_SERVICE_UNAVAILABLE    /**< Registration failed because LTE eMBMS is unavailable on
device   */
    };
};
```

B.3 IDL for DASH Streaming Service API

```
#include "EmbmsCommonTypes.idl"

module StreamingService
{

    //Forward Declaration
    interface  ILTEStreamingServiceCallback;


    /**
    * @name StreamingErrorCode
    * @brief List of the errors for streaming service
    */
    enum StreamingErrorCode
    {
        STREAMING_INVALID_SERVICE,        /**< Invalid service ID                    */
        STREAMING_UNKNOWN_ERROR           /**< Unknown error                         */
    };

    /**
    * @name StalledReasonCode
    * @brief List of the reasons for streaming service stalled notification
    */
    enum StalledReasonCode
    {
        RADIO_CONFLICT,         /**< Radio frequency conflict              */
        END_OF_SESSION,         /**< End of session schedule               */
        OUT_OF_COVERAGE,        /**< Out of EMBMS coverage                 */
        OUT_OF_SERVICE,         /**< Out of service                        */
        BEARER_UNAVAILABLE,     /**< Bearer not available                  */
        STALLED_UNKNOWN_REASON  /**< Unknown reason                        */
    };

    /**
    * @name RegisterStreamingAppData
    * @brief Streaming app registration information
    */
    struct  RegisterStreamingAppData
```

```
    {
        string    appId;                          /**< The application ID used during the
registration                        */
        any        platformSpecificAppContext;    /**< The platformSpecificAppContext
provides
                                                  a platform-specific app context
                                                  object to enable the API implementation to get
extra information
                                                  about the application.  */
        sequence<string> serviceClassList;        /**< List of service classes
*/
    };


    /**
    * @name StreamingServiceClassList
    * @brief ServiceClass information which the app is interested in. It is for
setStreamingServiceClassFilter API.
    */
    typedef sequence<string>  StreamingServiceClassList;



    /**
    * @name ServiceNameLang
    * @brief Name and language information
    */
    struct ServiceNameLang
    {
        string name;        /**< Name                  */
        string lang;        /**< Language              */
    };


    /**
    * @name StreamingServiceInfo
    * @brief Streaming service information
    */
    struct StreamingServiceInfo
    {
        sequence<ServiceNameLang> serviceNameList;    /**< List of Service name and language
*/
        string serviceClass;                          /**< Service class
*/
        string serviceId;                             /**< Service ID
*/
        string serviceLanguage;                       /**< Service language
*/

        EmbmsCommonTypes::ServiceAvailabilityType serviceBroadcastAvailability;  /**<
Service availability     */
        string mpdUri;                                /**< MPD URI used by DASH player
*/
        EmbmsCommonTypes::Date activeServicePeriodStartTime;    /**< The current/next active
file download service start time, when files
                                                                start being broadcast
over the air       */
        EmbmsCommonTypes::Date activeServicePeriodEndTime;      /**< The current/next active
file download service end time, when files
                                                                 stop being broadcast
over the air         */
        sequence<long> SAIList;                       /**< Servcie Area IDs based on current
location of the device*/
    };
```

```
    /**
    * @name StreamingServices
    * @brief List of streaming service info objects
    */
    typedef sequence<StreamingServiceInfo> StreamingServices;


    /**
    * @name StartStreamingServiceData
    * @brief Start streaming service information. It is used by StartStreamingService API.
    */
    struct  StartStreamingServiceData
    {
        string serviceId;        /**< Streaming service Id from StreamingServiceInfo
*/
    };

    /**
    * @name StopStreamingServiceData
    * @brief Stop streaming service information.
    *        It is used by the StopStreamingService API.
    */
    struct  StopStreamingServiceData
    {
        string serviceId;    /**< Streaming service ID from StreamingServiceInfo         */
    };


    /**
    * @name ServiceStartedNotification
    * @brief Streaming service started information. It is used by the
ServiceStartedNotification API.
    */
    struct ServiceStartedNotification
    {
        string serviceId;    /**< Streaming service Id from StreamingServiceInfo         */
    };

    /**
    * @name ServiceStoppedNotification
    * @brief Streaming service stopped information. It is used by the
ServiceStoppedNotification API.
    */
    struct ServiceStoppedNotification
    {
        string serviceId;    /**< Streaming service Id from StreamingServiceInfo         */
    };

    /**
    * @name StreamingServiceErrorNotification
    * @brief Streaming service error information. It is used by the
StreamingServiceErrorNotification API.
    */
    struct StreamingServiceErrorNotification
    {
        string serviceId;            /**< Streaming service Id from StreamingServiceInfo
*/
        StreamingErrorCode errorCode;      /**< Streaming service error Id
*/
        string errorMsg;      /**< error message                                        */
    };
```

```
    /**
    * @name ServiceStalledNotification
    * @brief Streaming service stalled information. It is used by the
ServiceStalledNotification API.
    */
    struct ServiceStalledNotification
    {
        string serviceId;            /**< Streaming service ID from StreamingServiceInfo
*/
        StalledReasonCode reason;    /**< Streaming service stalled reason ID
*/
    };


    /**
    * @name RegisterStreamingResponseNotification
    * @brief Streaming app registeration response information
    */
    struct RegisterStreamingResponseNotification
    {
        EmbmsCommonTypes::RegResponseCode value;  /**< Result of registeration value as
defined in RegResponseCode     */
        string message;         /**< message described the result
*/
    };

    interface ILTEStreamingService
    {

        /**
        @name getVersion
        @brief Retrieves the version of the current Streaming service interface
implementation
        @return Interface version
        **/
        string getVersion();

        /**
        @name registerStreamingApp
        @brief Application registers a callback listener with the EMBMS client
        @param[in] regInfo information required for application registration.
        @param[in] cb callback listener
        @see RegisterStreamingAppData
        @see registerStreamingResponse()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode registerStreamingApp(in RegisterStreamingAppData
regInfo, in ILTEStreamingServiceCallback callBack);


        /**
        @name deregisterStreamingApp
        @brief Application deregisters with the EMBMS client
        @pre Application calls register
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode deregisterStreamingApp();


        /**
        @name startStreamingService
        @brief Start download of segments of streaming service over broadcast
        @param[in] StartStreamingService Parameters for starting the streaming services API
```

```
        @pre Application is registered for streaming service
        @see StartStreamingServiceData
        @see serviceStarted()
        @see streamingServiceError()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode startStreamingService(in StartStreamingServiceData
serviceInfo);

        /**
        @name stopStreamingService
        @brief Stop download of segments of Streaming service over broadcast
        @param[in] StopDASHService Parameters for starting the streaming services API
        @pre Application is registered for DASH service
        @see serviceStopped()
        @see StopStreamingServiceData
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode stopStreamingService(in StopStreamingServiceData
serviceInfo);

        /**
        @name setStreamingServiceClassFilter
        @brief Application sets a filter on streaming services in which it is interested
        @param[in] serviceClassInfo List of service class filters requested by the
application
        @pre Application is registered successfully with streaming service
        @see serviceUpdate()
        @see getStreamingServices()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode setStreamingServiceClassFilter(in
StreamingServiceClassList serviceClassList);

        /**
        @name getStreamingServices
        @brief Retrieves the list of streaming services defined in the USD.
        List of services is filtered by the service class filter,
        if a filter has been set by the application.
        @param[out]  StreamingServices List of filtered streaming services
        @pre Application is registered for streaming service and received
streamingServiceListUpdate notification
        @see StreamingServices
        @see streamingServiceListUpdate()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode getStreamingServices(out StreamingServices services);
    };




    interface ILTEStreamingServiceCallback
    {

        /**
        @name registerStreamingResponse
        @brief The response to the application streaming service register API.
        @param  Notification Parameters for registering a streaming response
        @pre Application called registerStreamingApp
        @see RegisterStreamingResponseNotification
        @see registerStreamingApp()
        **/
```

```
        void registerStreamingResponse(in RegisterStreamingResponseNotification info);

        /**
        @name serviceStarted
        @brief Notification to application that streaming service is started and
        media player may be initialized for playback
        @param  Notification Parameters for service started notification.
        ServiceStartedNotification previously defined.
        @pre Application is registered for streaming service and called
startStreamingService
        @see ServiceStartedNotification
        **/
        void serviceStarted(in ServiceStartedNotification notification);

        /**
        @name serviceStopped
        @brief Notification to application that streaming service is stopped and
        media player may be stopped for playback
        @param Notification Parameters for service started notification
        @pre Application is registered for streaming service and called stopStreamingService
        @see ServiceStoppedNotification
        **/
        void serviceStopped(in  ServiceStoppedNotification notification);

        /**
        @name streamingServiceError
        @brief Notification to application when there is an error with broadcast download of
service
        @param Notification Parameters for service error notification
        @pre Application is registered for streaming service and called
startStreamingService
        @see StreamingServiceErrorNotification
        **/
        void streamingServiceError(in  StreamingServiceErrorNotification notification);

        /**
        @name serviceStalled
        @brief Notification to application when there is a temporary disruption of
        the broadcast download of service
        @param Notification Parameters for streaming service stalled notification
        @pre Application is registered for streaming service and called
startStreamingService
        @see ServiceStalledNotification
        **/
        void serviceStalled(in  ServiceStalledNotification notification);

        /**
        @name streamingServiceListUpdate
        @brief Notification to application on an update that is available for streaming
services.
        Update may be due to the received USD or the network configuration.
        @pre Application is registered for streaming service.
        @post call getStreamingServices()
        **/
        void streamingServiceListUpdate();

    };
};
```

B.4 IDL for MBMS RTP streaming delivery Service API

```
#include "EmbmsCommonTypes.idl"
```

```
module PacketService
{

    //Forward Declaration
    interface  ILTEPacketServiceCallback;


    /**
    * @name PacketErrorCode
    * @brief List of the errors for Packet service
    */
    enum PacketErrorCode
    {
        PACKET_INVALID_SERVICE,        /**< Invalid service ID                    */
        PACKET_UNKNOWN_ERROR           /**< Unknown error                         */
        MISSING_PARAMETER       /**< parameter is missing                  */
        NON_SUPPORTED_SERVICE_TYPE     /**< non supported service type            */
    };

    /**
    * @name StalledReasonCode
    * @brief List of the reasons for Packet service stalled notification
    */
    enum StalledReasonCode
    {
        RADIO_CONFLICT,         /**< Radio frequency conflict              */
        END_OF_SESSION,         /**< End of session schedule               */
        OUT_OF_COVERAGE,        /**< Out of EMBMS coverage                 */
        OUT_OF_SERVICE,         /**< Out of service                        */
        BEARER_UNAVAILABLE,     /**< Bearer not available                  */
        STALLED_UNKNOWN_REASON  /**< Unknown reason                        */
    };

    /**
    * @name ServiceType
    * @brief List of service types
    */
    enum ServiceType
    {
        RTP                     /**< Service Type for RTP                   */
       TRANSPARENT              /**< Service Type for TRANSPARENT           */
       TRANSPARENT-ROM     /**< Service Type for TRANSPARENT and ROM     */

    };


    /**
    * @name RegisterPacketAppData
    * @brief Packet Application registration information
    */
    struct  RegisterPacketAppData
    {

        string    ServiceType;             /** The requested service type)
        string    appId;                   /**< The application ID used during the
registration                      */
        any        platformSpecificAppContext;   /**< The platformSpecificAppContext
provides
                                            a platform-specific Application context
                                            object to enable the API implementation to get
extra information
                                            about the application.  */
        sequence<string> serviceClassList;    /**< List of service classes
*/
```

```
    };


    /**
    * @name PacketServiceClassList
    * @brief ServiceClass information which the Application is interested in. It is for
setPacketServiceClassFilter API.
    */
    typedef sequence<string>  PacketServiceClassList;



    /**
    * @name ServiceNameLang
    * @brief Name and language information
    */
    struct ServiceNameLang
    {
        string name;        /**< Name                 */
        string lang;        /**< Language             */
    };


    /**
    * @name PacketServiceInfo
    * @brief Packet service information
    */
    struct PacketServiceInfo
    {
        sequence<ServiceNameLang> serviceNameList;    /**< List of Service name and language
*/
        string serviceClass;                          /**< Service class */
        string serviceId;                             /**< Service ID  */
        string serviceLanguage;                       /**< Service language
*/
        EmbmsCommonTypes::ServiceAvailabilityType serviceBroadcastAvailability;  /**<
Service availability     */
        string sdpUri;                                /**< SDP URI used by Packet player
*/
        string interfaceName;                         /**< The network interface name used
by the Packet player to receive the data described in the SDP. */
        EmbmsCommonTypes::Date activeServicePeriodStartTime;    /**< The current/next active
Packet service start time, when Packet data
                                                                starts being broadcast
over the air       */
        EmbmsCommonTypes::Date activeServicePeriodEndTime;      /**< The current/next active
Packet service end time, when Packet data
                                                                 stops being broadcast
over the air         */
        sequence<long> SAIList;                       /**< Service Area IDs based on current
location of the device*/
    };


    /**
    * @name PacketServices
    * @brief List of Packet service info objects
    */
    typedef sequence<PacketServiceInfo> PacketServices;


    /**
    * @name StartPacketServiceData
    * @brief Start Packet service information. It is used by StartPacketService API.
```

```
    */
    struct  StartPacketServiceData
    {
        string serviceId;        /**< Streaming service Id from PacketServiceInfo         */
    };

    /**
    * @name StopPacketServiceData
    * @brief Stop Packet service information.
    *        It is used by the StopPacketService API.
    */
    struct  StopPacketServiceData
    {
        string serviceId;    /**< Streaming service ID from PacketServiceInfo         */
    };


    /**
    * @name ServiceStartedNotification
    * @brief Packet service started information. It is used by the
ServiceStartedNotification API.
    */
    struct ServiceStartedNotification
    {
        string serviceId;    /**< Streaming service Id from PacketServiceInfo         */
    };

    /**
    * @name ServiceStoppedNotification
    * @brief Packet service stopped information. It is used by the
ServiceStoppedNotification API.
    */
    struct ServiceStoppedNotification
    {
        string serviceId;    /**< Streaming service Id from PacketServiceInfo         */
    };

    /**
    * @name PacketServiceErrorNotification
    * @brief Packet service error information. It is used by the
PacketServiceErrorNotification API.
    */
    struct PacketServiceErrorNotification
    {
        string serviceId;            /**< Packet service Id from PacketServiceInfo
*/
        PacketErrorCode errorCode;      /**< Packet service error Id
*/
        string errorMsg;             /**< error message                              */
    };


    /**
    * @name ServiceStalledNotification
    * @brief Packet service stalled information. It is used by the
ServiceStalledNotification API.
    */
    struct ServiceStalledNotification
    {
        string serviceId;            /**< Packet service ID from PacketServiceInfo
*/
        StalledReasonCode reason;    /**< Packet service stalled reason ID               */
    };
```

```
    /**
    * @name RegisterPacketResponseNotification
    * @brief Packet Application registeration response information
    */
    struct RegisterPacketResponseNotification
    {
        EmbmsCommonTypes::RegResponseCode value;  /**< Result of registeration value as
defined in RegResponseCode     */
        string message;         /**< message described the result
*/
    };

    interface ILTEPacketService
    {

        /**
        @name getVersion
        @brief Retrieves the version of the current Packet service interface implementation
        @return Interface version
        **/
        string getVersion();

        /**
        @name registerPacketApp
        @brief Application registers a callback listener with the EMBMS client
        @param[in] regInfo information required for application registration.
        @param[in] cb callback listener
        @see RegisterPacketAppData
        @see registerPacketResponse()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode registerPacketApp(in RegisterPacketAppData regInfo, in
ILTEPacketServiceCallback callBack);


        /**
        @name deregisterPacketApp
        @brief Application deregisters with the EMBMS client
        @pre Application calls register
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode deregisterPacketApp();


        /**
        @name startPacketService
        @brief Start receiving Packet data over broadcast
        @param[in] StartPacketService Parameters for starting the Packet services API
        @pre Application is registered for Packet service
        @see StartPacketServiceData
        @see serviceStarted()
        @see packetServiceError()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode startPacketService(in StartPacketServiceData
serviceInfo);

        /**
        @name stopPacketService
        @brief Stop receiving Packet data over broadcast
        @param[in] StopPacketService Parameters for stoping the Packet services API
        @pre Application is registered for Packet service
        @see serviceStopped()
```

```
        @see StopPacketServiceData
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode stopPacketService(in StopPacketServiceData
serviceInfo);

        /**
        @name setPacketServiceClassFilter
        @brief Application sets a filter on Packet services in which it is interested
        @param[in] serviceClassInfo List of service class filters requested by the
application
        @pre Application is registered successfully with Packet service
        @see serviceUpdate()
        @see getPacketServices()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode setPacketServiceClassFilter(in PacketServiceClassList
serviceClassList);

        /**
        @name getPacketServices
        @brief Retrieves the list of Packet services defined in the USD.
        List of services is filtered by the service class filter,
        if a filter has been set by the application.
        @param[out]  PacketServices List of filtered Packet services
        @pre Application is registered for Packet service and received
packetServiceListUpdate notification
        @see PacketServices
        @see packetServiceListUpdate()
        @return ResultCode
        **/
        EmbmsCommonTypes::ResultCode getPacketServices(out PacketServices services);
    };


    interface ILTEPacketServiceCallback
    {

        /**
        @name registerPacketResponse
        @brief The response to the application Packet service register API.
        @param  Notification Parameters for registering a Packet response
        @pre Application called registerPacketApp
        @see RegisterPacketResponseNotification
        @see registerPacketApp()
        **/
        void registerPacketResponse(in RegisterPacketResponseNotification info);

        /**
        @name serviceStarted
        @brief Notification to application that Packet service is started and
        media player may be initialized for playback
        @param  Notification Parameters for service started notification.
        ServiceStartedNotification previously defined.
        @pre Application is registered for Packet service and called startPacketService
        @see ServiceStartedNotification
        **/
        void serviceStarted(in ServiceStartedNotification notification);

        /**
        @name serviceStopped
        @brief Notification to application that Packet service is stopped and
        media player may be stopped for playback
        @param Notification Parameters for service started notification
```

```
        @pre Application is registered for Packet service and called stopPacketService
        @see ServiceStoppedNotification
        **/
        void serviceStopped(in  ServiceStoppedNotification notification);

        /**
        @name packetServiceError
        @brief Notification to application when there is an error with broadcast download of
service
        @param Notification Parameters for service error notification
        @pre Application is registered for Packet service and called startPacketService
        @see PacketServiceErrorNotification
        **/
        void packetServiceError(in  PacketServiceErrorNotification notification);

        /**
        @name serviceStalled
        @brief Notification to application when there is a temporary disruption of
        the broadcast download of service
        @param Notification Parameters for Packet service stalled notification
        @pre Application is registered for Packet service and called startPacketService
        @see ServiceStalledNotification
        **/
        void serviceStalled(in  ServiceStalledNotification notification);

        /**
        @name packetServiceListUpdate
        @brief Notification to application on an update that is available for Packet
services.
        Update may be due to the received USD or the network configuration.
        @pre Application is registered for Packet service.
        @post call getPacketServices()
        **/
        void packetServiceListUpdate();

    };
};
```

Annex C (informative): IANA registration for MBMS URLs

C.1 General

This Annex provides the MBMS URL registration information that is referenced from the IANA registry at http://www.iana.org/ and follows the template per RFC 7595 [13], clause 7.4.

C.2 IANA Registration for Single Resource MBMS URL

Scheme name:

- mbms

Status:

- Permanent

Applications/protocols that use this scheme name:

- 3GPP file delivery over multicast-broadcast as specified in 3GPP TS 26.346 "Multimedia Broadcast/Multicast Service (MBMS)" and 3GPP TS 26.347 "MBMS URLs and APIs"

Contact:

3GPP Specifications Manager

3gppContact@etsi.org

+33 (0)492944200

Change controller:

3GPP SA WG4

References:

- 3GPP TS 26.347 "MBMS URLs and APIs"
- 3GPP TS 26.346 "Multimedia Broadcast/Multicast Service (MBMS)"
- both available at <http://www.3gpp.org/DynaReport/26-series.htm>

Security Considerations:

- The security considerations of section 7 of RFC 3986 apply. Maliciously constructed or other erroneous URIs may cause the MBMS API to search for non-existent services. Currently this is defined to be a local search, and not to perform a network interaction, but if a terminal were to contact network servers in an attempt to retrieve the service description, then providing such a URI to many user-agents simultaneously could cause network or server overload.

Annex D (informative): Service Name and Transport Protocol Port Number Registration

The following information is used to register the MBMS service with IANA according to RFC 6335 [16]:

Service Name	mbms
Transport Protocol	UDP
Assignee	3GPP SA4
Contact	Imed Bouazizi
Description	MBMS Delivery Service
Reference	3GPP TS 26.347
Port Number	Not defined

Annex E (informative): Implementation Guidelines for DASH over MBMS

E.1 General

This clause documents selected used cases and potential realizations for implementing DASH over MBMS, including hybrid services. The usage guidelines primarily focus on the information provided in the MBMS User Service Description and the application and usage of the information on the interface between the MBMS client and DASH client as shown in Figure E.1-1.

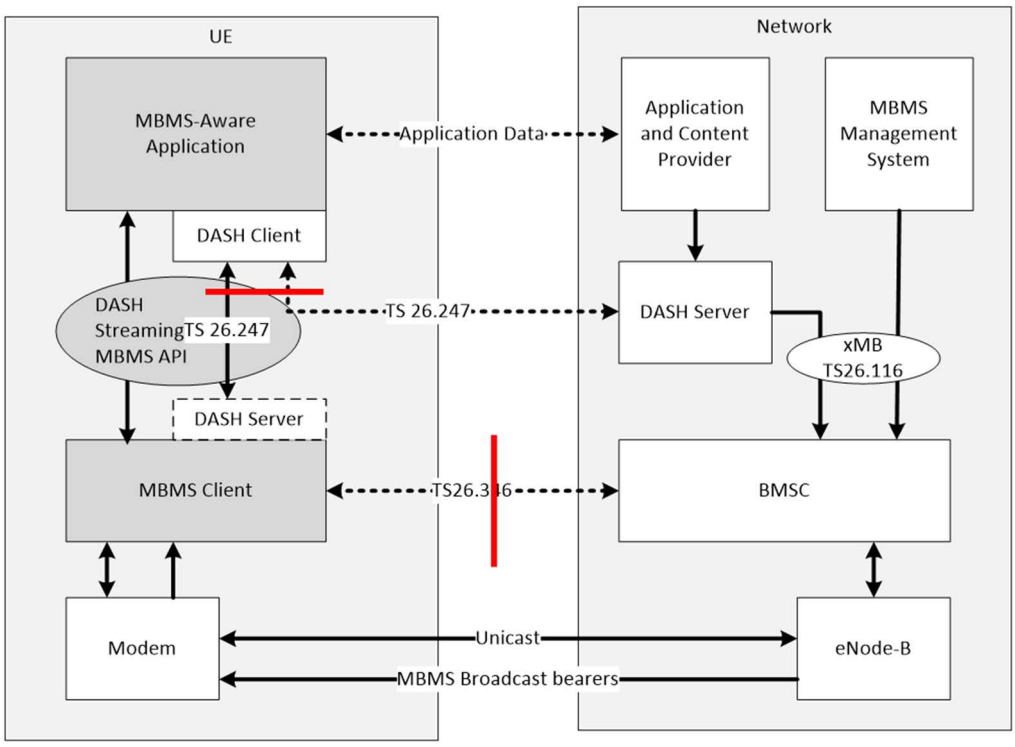

Figure E.1-1: Relevant interfaces and functions for the discussed use cases

E.2 Hybrid Service Offering with Unicast Fallback

E.2.1 Description

A sportscaster distributes a DASH-over MBMS service using the HEVC 720p HD signal over MBMS broadcast as it is aware that most of the devices are able to decode this content. The sportscasters' announcement track of the game in English is broadcast as well.

In addition the sportscaster delivers one or more program elements exclusively over the unicast bearer. As an example, an MBMS User Service comprising DASH-formatted streaming content for a live football game is broadcast in the Los Angeles metropolitan area, in which there is a significant Hispanic population. The sportscasters' announcement track of the game in English is broadcast, whereas the Spanish version of the announcement track is solely delivered over unicast.

In addition, the broadcaster has access to the content also in HDR and offers the content using the newly defined HDR FullHD profile as defined in Rel-15 of TS26.116 over unicast, offered in the same DASH-formatted service offering.

At the same time, the sportscaster also enables unicast fallback if the UE is out service coverage. For this purpose, a unified MPD is offered that contains all components of the service and provides also resources for broadcast distribution and unicast fallback.

The sportscaster desires that devices use the broadcast offered Representations in the following cases:

1) the client is in broadcast coverage and the user selects the HEVC 720p HD representation (coming over broadcast),

2) the client is in broadcast coverage and the user selects the English language (coming over broadcast)

3) the client is in broadcast coverage and the client is only capable to use HEVC 720p, but not the HDR FullHD profile.

E.2.2 Assumed MBMS User Service Description Signalling

In the User Service Description Signalling, the USDB contains the *r12:appService* element as part of the USD that contains the following information:

```
<r12:appService appServiceDescriptionURI="http://www.example.com/MPD2.mpd"
mimeType="application/dash+xml;profiles=urn:3GPP:PSS:profile:DASH10">
</r12:appService>
```

In addition, the MBMS receiver has access to *r12:broadcastAppService and r12:unicastAppService* element as follows:

```
<r12:broadcastAppService>
    <r12:basePattern>http://example.com/bc</r12:basePattern>
</r12:broadcastAppService>
<r12:unicastAppService>
    <r12:basePattern>http://example.com/uc</r12:basePattern>
</r12:unicastAppService>
<r15:supplementaryUnicastAppService>
    <r15:basePattern>http://example.com/suc</r15:basePattern>
</r15:supplemenaryUnicastAppService>
```

E.2.3 Assumed DASH MPD

It is assumed that the MPD referenced as http://www.example.com/MPD2.mpd in the User Service Description contains the following information:

```
<MPD
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mpeg:dash:schema:mpd:2011"
type="dynamic" minimumUpdatePeriod="PT10s"
timeShiftBufferDepth="PT600S"
minBufferTime="PT2S"
profiles="urn:3GPP:PSS:profile:DASH10"
publishTime="2014-10-17T17:17:05Z"
availabilityStartTime="2014-10-17T17:17:05Z">
<Location>http://www.example.com/MPD2.mpd</Location>
<Period id="1" start="PT0S">
<SegmentTemplate media="./$RepresentationID$/$Number$.m4s" initialization="$RepresentationID$-
init.mp4"/>
    <!– Video 720p -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.1.2.L93.B0" startWithSAP="1" maxWidth="1280"
maxHeight="720" frameRate="30" profile="urn:3GPP:video:op:h265-720p-HD">
     <BaseURL> http://example.com/uc</BaseURL>
      <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="v2048" bandwidth="2048000">
    <BaseURL> http://example.com/bc</BaseURL>
     </Representation>
      <Representation id="v1024" bandwidth="1024000"/>
      <Representation id="v512" bandwidth="512000"/>
      <Representation id="v128" bandwidth="128000"/>
    </AdaptationSet>
    <!– Video HDR -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.2.4.L113.B0" startWithSAP="1" maxWidth="1920"
maxHeight="1080" frameRate="30" profile="urn:3GPP:video:op:h265-Full-HD-HDR">
     <BaseURL> http://example.com/suc</BaseURL>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:MatrixCoefficients" value="9"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:TransferCharacteristics" value="16"/>
```

```
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:ColourPrimaries" value="9"/>
     <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="8M" bandwidth="8192000">
      <Representation id="6M" bandwidth="6144000"/>
      <Representation id="4M" bandwidth="4096000"/>
      <Representation id="2M" bandwidth="2048000"/>
    </AdaptationSet>
    <!– Audio English -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="en">
     <BaseURL> http://example.com/uc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
        <BaseURL> http://example.com/bc</BaseURL>
    </Representation>
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
    <!– Audio Spanish -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="es">
     <BaseURL> http://example.com/suc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
  </Period>
</MPD>
```

E.2.4 MBMS Client acting as DASH Server

It is assumed now that the MBMS client acts as DASH Server. In this case the MBMS client offers the MPD at a localhost URL as http://localhost/MPD2.mpd. The MBMS client acts as a partial proxy, i.e. only the MPD is proxied through the MBMS client.

In broadcast coverage, the MBMS client of offers the MPD such that only broadcast related Base URLs and supplemental Base URLs are available. It also rewrites the URLs and makes sure that the update period is frequent in order to make the DASH client aware of potential changes. The changed information is highlighted in bold and strikethrough compared to the MPD above:

```
<MPD
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mpeg:dash:schema:mpd:2011"
type="dynamic" minimumUpdatePeriod="0"
timeShiftBufferDepth="PT600S"
minBufferTime="PT2S"
profiles="urn:3GPP:PSS:profile:DASH10"
publishTime="2014-10-17T17:17:05Z"
availabilityStartTime="2014-10-17T17:17:05Z">
<Location>http://localhost/MPD2.mpd</Location>
<Period id="1" start="PT0S">
<SegmentTemplate media="./$RepresentationID$/$Number$.m4s" initialization="$RepresentationID$-
init.mp4"/>
    <!– Video 720p -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.1.2.L93.B0" startWithSAP="1" maxWidth="1280"
maxHeight="720" frameRate="30" profile="urn:3GPP:video:op:h265-720p-HD">
      <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="v2048" bandwidth="2048000">
    <BaseURL>http://localhost</BaseURL>
     </Representation>
    </AdaptationSet>
    <!– Video HDR -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.2.4.L113.B0" startWithSAP="1" maxWidth="1920"
maxHeight="1080" frameRate="30" profile="urn:3GPP:video:op:h265-Full-HD-HDR">
     <BaseURL> http://example.com/suc</BaseURL>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:MatrixCoefficients" value="9"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:TransferCharacteristics" value="16"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:ColourPrimaries" value="9"/>
     <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="8M" bandwidth="8192000">
      <Representation id="6M" bandwidth="6144000"/>
      <Representation id="4M" bandwidth="4096000"/>
      <Representation id="2M" bandwidth="2048000"/>
    </AdaptationSet>
    <!– Audio English -->
```

```
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="en">
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
        <BaseURL>http://localhost</BaseURL>
    </Representation>
    </AdaptationSet>
    <!– Audio Spanish -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="es">
     <BaseURL>http://example.com/suc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
  </Period>
</MPD>
```

Once the MBMS moves out of broadcast coverage the MPD is rewritten again as follows:

```
<MPD
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mpeg:dash:schema:mpd:2011"
type="dynamic" minimumUpdatePeriod="0"
timeShiftBufferDepth="PT600S"
minBufferTime="PT2S"
profiles="urn:3GPP:PSS:profile:DASH10"
publishTime="2014-10-17T17:17:05Z"
availabilityStartTime="2014-10-17T17:17:05Z">
<Location>http://localhost/MPD2.mpd</Location>
<Period id="1" start="PT0S">
<SegmentTemplate media="./$RepresentationID$/$Number$.m4s" initialization="$RepresentationID$-
init.mp4"/>
    <!– Video 720p -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.1.2.L93.B0" startWithSAP="1" maxWidth="1280"
maxHeight="720" frameRate="30" profile="urn:3GPP:video:op:h265-720p-HD">
     <BaseURL> http://example.com/uc</BaseURL>
      <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="v2048" bandwidth="2048000">
     </Representation>
      <Representation id="v1024" bandwidth="1024000"/>
      <Representation id="v512" bandwidth="512000"/>
      <Representation id="v128" bandwidth="128000"/>
    </AdaptationSet>
    <!– Video HDR -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.2.4.L113.B0" startWithSAP="1" maxWidth="1920"
maxHeight="1080" frameRate="30" profile="urn:3GPP:video:op:h265-Full-HD-HDR">
     <BaseURL> http://example.com/suc</BaseURL>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:MatrixCoefficients" value="9"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:TransferCharacteristics" value="16"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:ColourPrimaries" value="9"/>
     <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="8M" bandwidth="8192000">
      <Representation id="6M" bandwidth="6144000"/>
      <Representation id="4M" bandwidth="4096000"/>
      <Representation id="2M" bandwidth="2048000"/>
    </AdaptationSet>
    <!– Audio English -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="en">
     <BaseURL> http://example.com/uc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
    </Representation>
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
    <!– Audio Spanish -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="es">
     <BaseURL> http://example.com/suc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
  </Period>
</MPD>
```

E.2.5 MBMS Client acting as DANE

It is assumed now that the MBMS client acts as DANE. In this case the MBMS client offers the MPD at a localhost URL as http://localhost/MPD2.mpd. The MBMS client acts as a partial proxy, i.e. only the MPD is proxied through the MBMS client. The following MPD is provided, independent of the coverage situation.

```
<MPD
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mpeg:dash:schema:mpd:2011"
type="dynamic" minimumUpdatePeriod="0"
timeShiftBufferDepth="PT600S"
minBufferTime="PT2S"
profiles="urn:3GPP:PSS:profile:DASH10"
publishTime="2014-10-17T17:17:05Z"
availabilityStartTime="2014-10-17T17:17:05Z">
<Location>http://localhost/MPD2.mpd</Location>
<Period id="1" start="PT0S">
<SegmentTemplate media="./$RepresentationID$/$Number$.m4s" initialization="$RepresentationID$-
init.mp4"/>
    <!– Video 720p -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.1.2.L93.B0" startWithSAP="1" maxWidth="1280"
maxHeight="720" frameRate="30" profile="urn:3GPP:video:op:h265-720p-HD">
     <BaseURL> http://example.com/uc</BaseURL>
      <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="v2048" bandwidth="2048000">
    <BaseURL>http://localhost</BaseURL>
     </Representation>
      <Representation id="v1024" bandwidth="1024000"/>
      <Representation id="v512" bandwidth="512000"/>
      <Representation id="v128" bandwidth="128000"/>
    </AdaptationSet>
    <!– Video HDR -->
    <AdaptationSet mimeType="video/mp4" codecs="hvc1.2.4.L113.B0" startWithSAP="1" maxWidth="1920"
maxHeight="1080" frameRate="30" profile="urn:3GPP:video:op:h265-Full-HD-HDR">
     <BaseURL> http://example.com/suc</BaseURL>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:MatrixCoefficients" value="9"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:TransferCharacteristics" value="16"/>
     <EssentialDescriptor schemeIdUri="urn:mpeg:mpegB:cicp:ColourPrimaries" value="9"/>
     <SegmentTemplate timescale="30" duration="60"/>
      <Representation id="8M" bandwidth="8192000">
      <Representation id="6M" bandwidth="6144000"/>
      <Representation id="4M" bandwidth="4096000"/>
      <Representation id="2M" bandwidth="2048000"/>
    </AdaptationSet>
    <!– Audio English -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="en">
     <BaseURL> http://example.com/uc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
    <BaseURL>http://localhost</BaseURL>
     </Representation>
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
    <!– Audio Spanish -->
    <AdaptationSet mimeType="audio/mp4" codecs="mp4a.40.2" segmentAlignment="true" startWithSAP="1"
language="es">
     <BaseURL> http://example.com/suc</BaseURL>
      <SegmentTemplate timescale="20" duration="40"/>
      <Representation id="a128" bandwidth="128000">
      <Representation id="a64" bandwidth="64000">
    </AdaptationSet>
  </Period>
</MPD>
```

Along with every MPD request, an HTTP-URL is offered in the HTTP header as follows `SAND-header-field=MPEG-DASH-SAND:http://localhost/MBMS.sand`

In broadcast coverage, the following SAND messages is provided.

```
   <SAND>
    <Status baseURL="http://localhost/">
    <ResourceStatus status="cached"/>
    </Status>
```

```
 <Status baseURL="http://example.com/uc">
 <ResourceStatus status="unavailable"/>
 </Status>
 <Status baseURL="http://example.com/suc">
 <ResourceStatus status="available"/>
 </Status>
</SAND>
```

If the MBMS client is outside coverage, the following SAND messages is provided.

```
<SAND>
 <Status baseURL="http://localhost/">
 <ResourceStatus status="unavailable"/>
 </Status>
 <Status baseURL="http://example.com/uc">
 <ResourceStatus status="available"/>
 </Status>
 <Status baseURL="http://example.com/suc">
 <ResourceStatus status="available"/>
 </Status>
</SAND>
```

Annex F (informative): Change history

Date	TSG #	TSG Doc.	CR	Rev	Cat	Subject/Comment	New
2016-12	74	SP-160771				Presented to TSG SA#74 (for information)	1.0.0
2017-03	75	SP-170023				Presented to TSG SA#74 (for approval)	2.0.0
2017-03	75					Version 14.0.0	14.0.0
2017-06	76	SP-170327	0001	3		Service API for Transparent User Service	14.1.0
2017-06	76	SP-170319	0002	3		TRAPI: Multiple Corrections	14.1.0
2017-09	77	SP-170601	0003	6		Clarifications to DNS resolution in TRAPI	14.2.0
2018-06	80	SP-180272	0004	1		Support for SAND for MBMS	15.0.0
2018-12	82	SP-180971	0005	1	F	Support for SAND4M - Correction of Missing Text	15.1.0

History

Document history		
V15.0.0	July 2018	Publication
V15.1.0	April 2019	Publication