|  |  |
| --- | --- |
| 3GPP TR 26.857 V0.3.4 (2022-08) | |
| Technical Report | |
| 3rd Generation Partnership Project;  Technical Specification Group SA;  5G Media Service Enablers  (Release 18) | |
|  | |
|  |  |
|  | |
| The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP. The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented. This Specification is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification. Specifications and Reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices. | |

|  |
| --- |
|  |
| ***3GPP***  Postal address  3GPP support office address  650 Route des Lucioles - Sophia Antipolis  Valbonne - FRANCE  Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16  Internet  http://www.3gpp.org |
| ***Copyright Notification***  No part may be reproduced except as authorized by written permission. The copyright and the foregoing restriction extend to reproduction in all media.  © 2021, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).  All rights reserved.  UMTS™ is a Trade Mark of ETSI registered for the benefit of its members  3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners LTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners  GSM® and the GSM logo are registered and owned by the GSM Association |

Contents

Foreword 5

Introduction 6

1 Scope 7

2 References 7

3 Definitions of terms, symbols and abbreviations 8

3.1 Terms 8

3.2 Symbols 8

3.3 Abbreviations 8

4 Motivation for Media Service Enablers 9

4.1 General 9

4.2 Examples in 3GPP 10

4.2.1 MBMS Client 10

4.2.2 Media Session Handler in 5GMS 12

4.2.3 Media Player in 5GMS 13

4.2.4 3GPP TS 26.238 Flus sink capability discovery 14

4.2.5 SA6 Application Enabler Frameworks 15

4.3 External Specifications 15

4.3.1 General 15

4.3.2 W3C HTML-5 APIs for Media 16

4.3.2.1 W3C HTML-5 APIs 16

4.3.2.2 W3C Media Capabilities framework 16

4.3.3 Khronos OpenXR 17

4.3.4 MPEG Network-Based Media Processing function description 18

5 Considered MSE frameworks 18

5.1 General 18

5.2 MSE framework proposal #1 19

5.2.1 Architecture 19

5.2.2 MSE Specification 19

5.2.3 MSE implementation 20

5.2.4 Example 21

5.2.5 Benefits 23

5.3 MSE framework proposal #2 23

5.3.1 Overview 23

5.3.2 Architecture 23

5.3.3 Functions and reference points 25

5.3.4 Specification 26

5.3.5 Implementation support beyond specification 27

5.3.6 Style and documentation guidelines 28

5.3.7 Examples 29

5.3.7.1 Example 1: MBMS Client 29

5.3.7.2 Example 2: DASH Player 30

5.4 Discussion on different MSE framework proposals 30

6 MSE Specification Framework 31

6.1 Introduction 31

6.2 Initial MSE framework 31

6.2.1 General Concepts 31

6.2.2 MSE Reference Architecture 32

6.2.3 Functions and reference points 33

6.3 MSE specification Template 33

6.4 Beyond the MSE Specification – guidelines, tests and reference implementations 36

7 Writing MSE Specifications: Style Guides and Tools 37

8 Potentially Relevant 5G Media Service Enablers 39

9 Conclusions and Recommendations 39

Annex A (informative): Details on Tools and Templates 39

Annex <X> (informative): Change history 40

# Foreword

This Technical Report 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.

In the present document, modal verbs have the following meanings:

**shall** indicates a mandatory requirement to do something

**shall not** indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

**should** indicates a recommendation to do something

**should not** indicates a recommendation not to do something

**may** indicates permission to do something

**need not** indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

**can** indicates that something is possible

**cannot** indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

**will** indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

**will not** indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

**might** indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

**might not** indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

**is** (or any other verb in the indicative mood) indicates a statement of fact

**is not** (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

# Introduction

In recent studies and specification work, it was identified that 5G Media functions and 5G System functions need to be made attractive for third-party applications, in particular those that include media delivery. Hence, it is important that these functions are accessible to third-party applications independent of a 3GPP service. For this purpose, it is considered to introduce normative specifications in 3GPP SA4 that are

- more than just a core functionality, e.g. a codec, without any connection to a service or application

- less than a full service that includes all aspects of session establishment, delivery, codecs, rendering and a full user experience

Such new specifications are referred to 5G "Media Service Enablers".

# 1 Scope

The present document introduces and defines the concept of Media Service Enablers which includes among others:

* Definition of the principal properties of Media Service Enablers.
* Definition of minimum and typical functionalities of Media Service Enablers.
* Definition of a specification template for Media Service Enablers.
* Identification of possibly relevant stage-2 and stage-3 work for Media Service Enablers.
* Collection of a set of initially relevant Media Service Enablers for normative work.

# 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] ISO/IEC 23090-8:2020: "Information technology — Coded representation of immersive media — Part 8: Network based media processing".

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

[4] 3GPP TS 26.347: "Multimedia Broadcast/Multicast Service (MBMS); Application Programming Interface and URL".

[5] 3GPP TS 26.479, "UE MBMS APIs for Mission Critical Services".[D] 3GPP TS 26.501, "5G Media Streaming (5GMS); General description and architecture".

[6] 3GPP TS 26.511, "5G Media Streaming (5GMS); Profiles, codecs and formats".

[7] 3GPP TS 26.512, "5G Media Streaming (5GMS); Protocols".

[8] 3GPP TS 23.434: "Service Enabler Architecture Layer for Verticals (SEAL); Functional architecture and information flows".

[9] The Khronos Group, "The OpenXR Specification",  
<https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html>

[10] W3C: "Media Capabilities", Working Draft, 3 August 2022,  
<https://www.w3.org/TR/media-capabilities/>

[11] 3GPP TS 26.238: "Uplink Streaming".

[12] Mission Critical Open Platform, <https://www.mcopenplatform.org/>

[13] 3GPP TS 26.501, "5G Media Streaming Architecture".

[14] ISO/IEC 23000-19: Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media.

[15] ISO/IEC 23009-1: " Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats".

[16] 3GPP TS 29.501, "5G System; Principles and Guidelines for Services Definition; Stage 3".

# 3 Definitions of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the terms given in 3GPP 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 3GPP TR 21.905 [1].

**eXtended Reality:** A continuum of real-and-virtual combined environments generated by computers through human-machine interaction, including the technologies associated with Virtual Reality (VR), Augmented Reality (AR) and Mixed Reality (MR).

## 3.2 Symbols

For the purposes of the present document, the following symbols apply:

Editor’s Note: needs to be completed

<symbol> <Explanation>

## 3.3 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP 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 3GPP TR 21.905 [1].

Editor’s Note: needs to be completed

API

AR Augmented Reality

ASCII

CMAF

CTA

CTS

DASH

EME

FLUS

HLS

HTML

HTTP

IDL

JSON

KPI

MBMS

MBS

MCA

MDD

MPD

MPEG

MR Mixed Reality

MRP

MSC

MSE Media Service Enabler

MSH

MSK

NBMP

RTP

SDK

SEAL Service Enabler Architecture Layer for Verticals

UDP

UPF

URL

WAVE

VAL Vertical Application Layer

VR Virtual Reality

XR eXtended Reality

YAML

# 4 Motivation for Media Service Enablers

## 4.1 General

In recent studies and specification work, it was identified that 5G Media functions and 5G System functions need to be made attractive for third-party applications, in particular those that include media delivery. Examples for such approaches are MBMS or 5G Media Streaming. Hence, it is important that these functions are accessible to third-party applications independent of a 3GPP service. For this purpose, it is considered to introduce normative specifications in 3GPP that are:

* More than just a core functionality, e.g. a codec, without any connection to a service or application.
* Less than a full service that includes all aspects of session establishment, delivery, codecs, rendering and a full user experience.

The specification should also not only address a pure textual description but provide additional functionalities such as test and validation tools.

Several examples of specifications at least partially addressing such needs are provided in the remainder of this clause, both 3GPP internal specifications in clause 4.2 and external specification in clause 4.3.

## 4.2 Examples in 3GPP

### 4.2.1 MBMS Client

An example for the definition of an API-centric component in a 3GPP specification is one that serves the MBMS Client. The detailed procedures of the MBMS Client are defined in 3GPP TS 26.346 [3] and TS 26.347 [4] according to Figure 4.2.1-1.



Figure 4.2.1-1: MBMS Client – Application and Network reference Points an APIs

In particular, TS 26.347 defines the following aspects:

1. A set of service APIs for different application user services. The definition provides the ability to independently develop MBMS-Aware Applications and MBMS Client implementations, even for different operating systems and execution environments, but relies on the service APIs to communicate with the MBMS Client and to make use of the MBMS functionalities. These APIs are referred to as MBMS-API-C.

2. A set of interface options between the MBMS Client and the application to support the transfer of user data. The primary focus is on the communication through network interfaces, for example the usage of IP sockets or HTTP-based requests. These APIs are referred to as MBMS-API-U.

Additionally, For Mission Critical (MC) purposes and direct access to MBMS bearer contents, an integration API is specified by the Mission Critical Open Platform [12]. 3GPP also specifies the MC MBMS API in TS 26.479 [5] based on the same objective.

The APIs defined in TS 26.347 address the following aspects:

- A *client state model* in relation to the application. Examples for state are IDLE, REGISTERED, ACTIVE, etc. State changes may occur through MBMS-API-C or by information received through the network interface.

- A set of *client internal parameters* that are changed based on either configuration or API calls through MBMS-API-C or by information received through the network interface.

- A *reference description* of the operation of the MBMS client in different states, based on through MBMS-API-C or by information received through the network interface

- Different *methods* that allow the application to communicate with the MBMS client. For each method, the following information is provided:

i) A *high-level description* of the method.

ii) An example *call flow* illustrating usage of the method.

iii) A list of input and output *parameters* that are exchanged as part of the method invocation.

iv) A *description of the usage* of the method by the application.

v) the MBMS Client actions in response to the invocation of the method, including pre- and post-conditions.

The equivalent Android APIs for MBMS-API-C are defined in the developer framework of Android:

* Download Session: <https://developer.android.com/reference/android/telephony/MbmsDownloadSession>
* Group Call Session: <https://developer.android.com/reference/android/telephony/MbmsGroupCallSession>
* Streaming Session: <https://developer.android.com/reference/android/telephony/MbmsStreamingSession>
* MBMS API documentation: https://developer.android.com/reference/android/telephony/mbms/package-summary

Finally, TS 26.347 also defines interfaces between the MBMS Client and the application for data exchanges. While the MBMS-API-C provides all methods to find and establish these interfaces, MBMS-API-U provides requirements on the data interfaces, for example for copying files, for requesting files through HTTP, for using specific methods based on an application such as DASH or HLS, or for accessing interfaces that provide RTP packets, UDP datagrams or packet data.

An example usage of the abovementioned Android APIs to support accessing MBMS services through Mission Critical functions is provided as follows using a reception feature activation:

Listing

4.2.1‑1

private MbmsGroupCallSessionCallback groupCallSessionCallback;  
private MbmsGroupCallSession mbmsGroupCallSession;  
private String mbmsInterfaceName;  
*/\*\*  
 \* Activate MBMS reception  
 \*/*public void enableMBMS() {  
  
 groupCallSessionCallback = new MbmsGroupCallSessionCallback() {  
  
 @Override  
 public void onServiceInterfaceAvailable(@NonNull String interfaceName, int index) {  
 Log.*e*(*TAG*, "service interface for MBMS Reception " + interfaceName);  
 mbmsInterfaceName = interfaceName;  
 }  
  
 };  
  
 //Enabling MBMS reception  
 mbmsGroupCallSession = MbmsGroupCallSession.*create*(this.getApplicationContext(), 1, this.getMainExecutor(), groupCallSessionCallback);  
   
}

Reception of data from an MBMS bearer is triggered as follows:

Listing 4.2.1‑2

private GroupCall groupCall;

*/\*\*  
 \* Starting the reception of a MBMS bearer  
 \** ***@param*** *tmgi the Temporary Multicast Group Identifier of the MBMS Bearer  
 \*/*public void startReceptionMBMSBearer(long tmgi) {  
 GroupCallCallback myCallBack = new GroupCallCallback() {  
 @Override  
 public void onGroupCallStateChanged(int state, int reason) {  
 switch (state) {  
 case GroupCall.*STATE\_STARTED*:  
 Log.*i*(*TAG*, "MBMS bearer reception is started");  
 break;  
  
 case GroupCall.*STATE\_STALLED*:  
 Log.*e*(*TAG*, "onGroupCallStateChanged: stalled reason " + reason);  
 break;  
  
 case GroupCall.*STATE\_STOPPED*:  
 Log.*e*(*TAG*, "onGroupCallStateChanged: stopped reason " + reason);  
 break;  
 } } };  
 //List of Service Area Identifiers and frequencies, may be left empty  
 List<Integer> sais = new ArrayList<Integer>();  
 List<Integer> frequencies = new ArrayList<Integer>();  
 groupCall = mbmsGroupCallSession.startGroupCall(tmgi, sais, frequencies, *executor*, myCallBack);

Finally, the multicast packet data is accessed by the following execution:

Listing 4.2.1‑3

*/\*\*  
 \* Access to the multicast IP packets   
 \** ***@param*** *multicastAddress String representation of the multicast IP address to join  
 \** ***@param*** *destinationPort destination port  
 \** ***@throws*** *Exception  
 \*/*public void receive(String multicastAddress, int destinationPort) throws Exception {  
 NetworkInterface ni = NetworkInterface.*getByName*(mInterfaceName);  
  
 //open a multicast socket  
 MulticastSocket mSocket = new MulticastSocket(destinationPort);  
 SocketAddress socketAddress =  
 new InetSocketAddress(multicastAddress, destinationPort);  
  
 //join the multicast group on a given network interface  
 mSocket.joinGroup(socketAddress, ni);  
  
 while (true) {  
 byte[] buf = new byte[1500];  
 DatagramPacket recv = new DatagramPacket(buf, buf.length);  
 mSocket.receive(recv);  
  
 //*TODO process the received datagram* }  
}

### 4.2.2 Media Session Handler in 5GMS

Another example enabling function relevant to 5G media delivery is the Media Session Handler defined in TS 26.501 [13] (stage-2) and TS 26.512 [6] (stage-3). The Media Session Handler is a function on the UE that communicates with the 5GMSd AF in order to establish, control and support the delivery of a media session, and may perform additional functions such as the collection and reporting of consumption and QoE metrics. The Media Session Handler exposes APIs that can be used by the 5GMSd-Aware Application. An overview is provided in Figure 4.2.2-1.



Figure 4.2.2-1: Media Session Handler– Application and Network reference Points an APIs  
(Reproduced from 3GPP TS 26.512 [6])

The Media Session Handler deals with three sets of APIs and reference points:

- *M5d (Media Session Handling API):* APIs exposed by a 5GMSd AF to the Media Session Handler for media session handling, control, reporting and assistance that also include appropriate security mechanisms, e.g. authorization and authentication.

- *M6d (UE Media Session Handling APIs):* APIs exposed by a Media Session Handler to the Media Player for client-internal communication and exposed to the 5GMSd-Aware Application enabling it to make use of 5GMS functions.

NOTE: The M6d APIs are not yet fully specified in TS 26.512 [6].

- *M7d (UE Media Player APIs):* APIs exposed by a Media Player to the 5GMSd-Aware Application and Media Session Handler to make use of the Media Player.The APIs for M6d and M7d are defined in an abstract manner at this stage.

### 4.2.3 Media Player in 5GMS

Downlink 5G Media Streaming specifies the use of segment formats that are based on the Common Media Application Format (CMAF) in ISO/IEC 23000-19 [14]. By using this format, 5G Media Streaming is compatible with a broad set of segment-based streaming protocols including Dynamic Streaming over HTTP (DASH) and HTTP Live Streaming (HLS). For example, ISO/IEC 23009-1 [15] defines a detailed DASH profile for delivering CMAF content within a DASH Media Presentation using a converged format for segmented media content.



Figure 4.2.3-1: Media Playback in 5G Media Downlink Streaming Architecture  
(reproduced from 3GPP TS 26.511 [7])

According to TS 26.511 [7], TS 26.512 [6] and Figure 4.2.3-1 above, the Media Player is further decomposed into an Access Client and a Media Playback Platform. Several APIs are identified for the Media Player:

- *M4d (Media Streaming APIs):* APIs exposed by a 5GMSd AS to the Media Player to stream media content.

*- M6d (UE Media Session Handling APIs):* APIs exposed by a Media Session Handler to the Media Player for client-internal communication and exposed to the 5GMSd-Aware Application enabling it to make use of 5GMS functions.

*- M7d (UE Media Player APIs):* APIs exposed by a Media Player to the 5GMSd-Aware Application and Media Session Handler to make use of the Media Player.

- A set of internal *Media Player APIs* that deals with providing accessed data to the Media Playback Platform. These closely follow the W3C APIs for HTML-5 based media playback and the Media Source Extensions.

Most relevant in the discussion is the M7d API provided by the Access Client (see clause 13 of TS 26.512 [6]) defining:

1) Methods to interact with the Access Client of the Media Player,

2) Notification and Error Events sent to the Media Session Handler and 5GMSd-Aware Application,

3) Configuration and Settings methods,

4) Status Information.

The initial API has largely been designed based on the dash.js API documented here: <http://cdn.dashjs.org/latest/jsdoc>, but they are abstract.

For the Media Player, different states are defined, depending on actions received from any of the APIs.

### 4.2.4 FLUS sink capability discovery

3GPP TS 26.238 [11] defines a set of protocols for uplink media streaming. This specification includes a method for describing the processing capabilities of the entity (known as FLUS sink) that receives the uplink stream. In this specification, these capabilities are described as a list. Each entry in the list includes a scheme identifier, the location for the description of the scheme, and a URL where the specific capability can be accessed. The FLUS sink capabilities description can be retrieved from the sink or it can be found in a sink directory.

The advantage of the FLUS sink capabilities description is its simplicity. However, since each item in the capabilities list has its own scheme, it does not provide much interoperability for describing the available functions and their detailed features, since each function defines its own scheme for describing its capabilities.

### 4.2.5 Service Enabler Architecture Layer (SEAL) for Verticals

SA6 defines several Application Frameworks, for example Service Enabler Architecture Layer for Verticals (SEAL). TS 23.434 [8] specifies the functional architecture of the Service Enabler Architecture Layer (SEAL) and the procedures, information flows and APIs for each service within SEAL in order to support vertical applications over the 3GPP system. To ensure efficient use and deployment of vertical applications over 3GPP systems [8] includes the group management, configuration management, location management, identity management, key management and network resource management. Figure 4.2.5-1 illustrates the generic on-network functional model for SEAL.



Figure 4.2.5-1: Generic on-network functional model for SEAL (see TS 23.434, Figure 6.2-1)

In the vertical application layer (VAL), the *VAL client* communicates with the *VAL server* over reference point VAL-UU. This supports both unicast and multicast delivery modes, but is otherwise out of scope of SEAL.

The SEAL functional entities on the UE and the server are grouped into *SEAL client(s)* and *SEAL server(s)* respectively. The SEAL consists of a common set of services (e.g. group management, location management) and reference points. The SEAL offers its services to the vertical application layer (VAL). The functionalities and reference points of the vertical application layer are out of scope of SEAL.

- Each SEAL client communicates with its SEAL server over reference point SEAL-UU.

- The SEAL client provides the service enabler layer support functions to the VAL client over reference point SEAL-C.

- Each VAL server communicates with its SEAL server over reference point SEAL-S.

A SEAL server may communicate with the underlying *3GPP network system* using the *Network interfaces* provided by the 5G System (labelled *3GPP network system*). The specific SEAL client(s) and the SEAL server(s), along with their specific instantiations of reference point SEAL-UU and the specific network interfaces of the 3GPP network system used, are described in the respective on-network functional model for each SEAL service.

For each such service, TS 23.434 [8] defines the functional model, procedures and information flows, as well as the APIs. The focus in [8] is on stage-2; detailed stage-3 is not defined.

## 4.3 External Specifications

### 4.3.1 General

This clause provides an overview of existing specifications that provide specifications similar to what is envisioned as Media Service Enabler specifications.

### 4.3.2 W3C HTML-5 APIs for Media

#### 4.3.2.1 W3C HTML-5 APIs

EDITOR#S Note.

W3C way

Significant testing environment

2 interoperable implementations

<https://www.w3.org/developers/tools/>

https://github.com/w3c/respec

CTA WAVE 5003 Device Playback specification

Diagram

Description automatically generated

#### 4.3.2.2 W3C Media Capabilities framework

The W3C Media Capabilities API [10] is designed to query a W3C user agent (such as a web browser) with regard to its media decoding and encoding capabilities. The intent is to provide a replacement for the HTML5 canPlayType() method and the Media Source Extension isTypeSupported() method that more accurately reflects the user agent’s media encoding and decoding capabilities.

The API supports decoding from a file, from Media Source Extension (MSE), or from WebRTC, and also encoding to media chunks that can be stored in a file or transmitted as a WebRTC stream.

- The video and audio configuration objects define the general characteristics of the video and audio configuration.

- The key system configuration object defines the key configuration for the encrypted media (EME).

For a given encoding or decoding configuration, the API returns a binary flag for each of the following values: supported, smooth, or power-efficient. It seems the logic is very similar to canPlayType(), but the configuration can be described in more detail and the result is more accurate.

The specification is characterised as follows:

1. It limits the expression of capabilities to video and audio encoding/decoding and key configuration for decryption. It doesn’t support other media types such as subtitles, images, or graphics.

2. It doesn’t define a general language to define the capabilities of various media functions other than video/audio decoding.

3. It seems various encoding configurations (e.g. motion estimation, bit rate control modes, number of reference frames, etc) cannot be described or set as the desired configuration.

4. The API for querying the functionality i still based on a binary pass/fail result (similar to canPlayType()), presumably to reduce the risk of fingerprinting the device, and therefore, it is not possible to get an expressive list of supported features.

### 4.3.3 Khronos OpenXR

OpenXR [9] is an API (Application Programming Interface) for XR applications. It sits between an application and an in-process or out-of-process "XR runtime system" (just "runtime" hereafter). The runtime may handle such functionality as frame composition, peripheral management, and raw tracking information.

The OpenXR specification is intended to satisfy the needs of both programmers and runtime implementors:

* To the application programmer, OpenXR is a set of functions that interface with a runtime to perform commonly required operations such as accessing controller/peripheral state, getting current and/or predicted tracking positions, and submitting rendered frames.
* To the runtime implementor, OpenXR is a set of functions that control the operation of the XR system and establishes the lifecycle of an XR application.

However, the specification does not necessarily provide a model for implementation. A runtime implementation is expected to produce results conforming with those produced by the specified methods, but may carry out particular procedures in ways that are more efficient than the one specified.

A few fundamentals on API definitions:

* Version numbers 64 bit major, minor, patch.
* Threading, multiprocessing, runtime.
* Extensions.
* API Layering, which means that a user or application may insert API layers between the application and the runtime implementation.
* Return codes.
* Handles and Object Handling.
* Timing.
* Colors.
* Coordinate Systems.
* Events.

The specification is also supported by API Reference Pages [9] which are generated by automatic extraction from the specification source and document commands, object handles, structures, enumerations, flags, other types and all extensions.

For each command, the following information is provided:

* Name.
* C function call specification.
* Parameter descriptions.
* Valid usage.
* Return codes.
* Reference to OpenXR specification.

OpenXR is an extensible API and the optional functions are call extensions. Since the extensions are options, only a subset of runtimes may implement a particular extension. Therefore, an application may first query which extensions are available from its underlying runtime. Open XR permits extensions to be defined by OpenXR or by external parties. The function [xrEnumerateInstanceExtensionProperties](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html" \l "xrEnumerateInstanceExtensionProperties) [9] provides the list of available extensions in the current runtime. Each extension is identified by a unique name and an extension version, and by the type of the extension.

For the specification itself, the document sources are marked up in *Asciidoctor* format (https://asciidoctor.org/). *Asciidoctor* and related toolchain components are used to generate the output documents. A full overview of how the specification is developed is provided here: <https://github.com/KhronosGroup/OpenXR-Docs/blob/main/specification/README.md>.

Khronos has also released a Conformance Test Suite for OpenXR, published the tests as open source software on GitHub (https://github.com/KhronosGroup/OpenXR-CTS), and launched the OpenXR 1.0 Adopters Program so that implementations can be officially conformant for the first time. Any OpenXR implementer – Khronos member or not – is welcome to become an OpenXR Adopter and submit conformance test results for Working Group review and approval. If their implementation is approved as conformant, they will be able to use the OpenXR trademark on their implementation.

### 4.3.4 MPEG Network-Based Media Processing (NBMP) media function description

ISO/IEC 23090-8 [2] is a specification for describing media functions and workflows as microservices running in the cloud. The specification defines the data structures for describing workflows and methods for managing them. The framework enables dynamic creation of media processing pipelines, as well as access to processed media data and metadata in real time or in a deferred way. The media and metadata formats used between the media source, workflow manager, and media processing entities in a media processing pipeline are also specified.

Each function in a media processing pipeline is specified using a JSON object called a *function description* that describes the functionality, the inputs, outputs, processing characteristics, and configuration parameters, as well as the processing requirements. The function description can also describe events, and even how to set up notifications, reporting, and monitoring for that function if it is deployed as a microservice. More complex functions can be described also as a composite of simpler functions. Finally, the function description JSON object can also be used programmatically to instantiate a task or microservice that performs the function it describes.

The NBMP function description has the advantage of describing being able to describe a function with identifiers as well as a detailed description of a function. The optional parameters of an implementation of a generic function can be described using this approach.

# 5 Considered MSE frameworks

## 5.1 General

This clause collects some proposed and considered MSE frameworks. A discussion on the different framework proposals is provided in clause 5.4.

## 5.2 MSE framework proposal #1

### 5.2.1 Architecture

Figure 4.4.1.1-1 shows a possible framework for Media Service Enablers. The MSE framework consists of two parts: the *MSE specification* (on the left of the figure) and the *MSE implementation* (on the right).

**MSE Specification**

(Platform-dependent) **MSE SDK**

Media interfaces

Control interfaces

Configuration API

**MSE SDK abstraction**

Media interfaces

Control interfaces

Configuration API Abstraction

**MSE Service**

Media interfaces

Control interfaces

Configuration API

*Specification*

*Implementation*

MSE Description Document

Media specification

MSE Configuration API

Service API

*Platform-independent*

*Platform-dependent*

**(c)**

**(b)**

**(d)**

**(a)**

Figure 5.2.1-1. Media Service Enablers Framework

### 5.2.2 MSE Specification

An MSE Specification (a) defines:

1. *Media aspects*

a. Functional description of the MSE including the mandatory and optional features.

b. The control interfaces such as provisioning, authentication that is used by the application, and other functions to interact with this MSE.

c. The media interfaces that includes all inputs and outputs format and protocols.

d. Network interface including system and radio network.

e. Event, notifications, reporting, and monitoring.

f. Error handling.

2. *MSE Configuration*

a. An *MSE Description Document (MDD)* that describes an implementation’s functional support in a standardised way, including:

1. Functions supported by an MSE implementation and their configuration parameters.

2. Optionally the performance/cost metrics for the different features/options.

b. An *MSE Configuration API (MCA)* abstraction for:

1. Optionally retrieving the MSE Description Document.

2. Configuring the MSE instantiation.

3. Optionally retrieving the state and status of the MSE instantiation.

c. A service API for the MSE Configuration API.

Media aspects (1) are usually covered by SA4 specifications. However, the MSE Configuration (2) is absent from current SA4 specifications and is what the MSE Specification adds. The value of this is that, for any SDK or service that is conforming to the MSE specification, a description of the features and their configuration parameters can be described using a standard document format. Furthermore, this description can be retrieved through the configuration API if supported by the implementation. Additionally, the external function or service can set a specific configuration for running that SDK. Furthermore, the state and status of the running SDK can be retrieved at any time.

The language and syntax of the MSE Description Document and the general framework of the MSE Configuration API can be defined uniformly for all SA4 Media Service Enabler specifications and only specific codepoints are defined in that specification. An external function or application understanding the MSE Description Document syntax, as well as supporting the MSE Configuration API, can retrieve the information from an MSE implementation. If it recognizes the MSE Specification identifier, it can parse and process the MSE Description Document and its configuration parameters.

An example of an MSE Description Document can be found in ISO/IEC 23090-8 [2]. The function description document is a JSON document that describes the functionalities and features that a function provides as well as its configuration parameters.

### 5.2.3 MSE implementation

An MSE implementation may consist of up to three aspects:

1. The MSE SDK abstraction (c), an abstract SDK definition intended to be realized as a Software Development Kit, which includes the followings:

i. Media aspects conforming to the MSE specification.

ii. MSE Description Document and MSE Configuration API.

2. The MSE SDK instantiation (d) which is an SDK implementation in a specific environment and conforms to the following:

i. Media aspects conforming to the MSE Specification.

ii. MSE Description Document and a specific implementation of the MSE Configuration API.

3. The MSE service (b) which is the MSE implementation as a service, i.e with APIs that are platform-independent (such as web-based APIs) and conforms to the following:

i. Media aspects conforming to the MSE Specification.

ii. MSE Description Document and a platform-independent implementation of the MSE Configuration API.

As shown in Figure 4.4.1.1-1, while the MSE SDK abstraction and the MSE Service are platform-independent, the MSE SDK is an instantiation of the MSE SDK abstraction for a specific platform/environment.

An MSE Specification does not required to include all three aspects. For instance, if an MSE is only intended to be realized as a software development kit, then its specification would include specifications for the SDK abstraction and one or more SDK instantiation.

Note that in the cases of MSE SDK abstract SDK (c) and MSK SDK (d), the MDD may not be retrievable through the MSE configuration APIs. In these cases, MDD is a side document, describing the features supported by the SDK.

Table 5.2.4-1 summarizes the above features.

Table 5.2.4-1. Summary of MSE features for various components of Figure 5.2.4-1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Specification  (a) | MSE service  (b) | MSE SDK abstract  (c) | MSE SDK  (d) |
| MSE Description Document (MDD) | Describes the mandatory and optional features in a standard way. | Describes features implemented and configuration options using MDD.  This document can be retrieved using MCA. | Describes features implemented and configuration options using MDD.  This might be a side document. | Describes features implemented and configuration options using MDD.  This might be a side document. |
| MSE Configuration API (MCA) | The API abstraction describing how to configure the MSE. | A platform-independent API for MSE configuration. | An abstract API for MSE configuration. | An API instance for MSE configuration. |

### 5.2.4 Example

As shown in figure 5.2.1-1, the MSE Specification can be deployed in two different ways: as an SDK for running on devices or as a microservice running on an Application Server. To demonstrate converting an existing 3GPP specification to an MSE specification, we use the 5GMS Media Session Handler defined in TS 26.501 [D], shown in figure 5.2.4-1.



Figure 5.2.4-1. Media Session Handler as defined in 26.501



Figure 5.2.4-2. Media Session Handler as MSE SDK abstraction, MSE SDK instantiations, and MSE service

The MSE Specification for the Media Session Handler (MSH) shown in Figure 5.2.4-2 describes the following:

1. Media aspects:

a. Functional description of:

i. Service Access Information.

ii. Consumption Reporting.

iii. Metrics Reporting.

iv. Dynamic policies.

v. Network Assistance.

b. M5d, M6d, M7d API definitions:

i. M5d as is already defined.

ii. M6d and M7d as abstract APIs.

iii. M6d and M7d as service APIs.

2. MSE Configuration

a. An MSE Description Document which describes:

i. An identifier that shows this MSE conforms to (1).

ii. Optional features of (1a) and (1b) with their configuration parameters.

iii. Optionally the performance/cost metrics for the different features/options.

b. Abstract API definitions for:

i. Retrieving the MSE Description Document (2a).

ii. Configuring the MSE instantiation.

iii. Retrieving the state and status of the MSE instantiation.

c. A service API for the abstract API (2b).

And MSE SDK implementation of the above specification for Android should support the following:

3. Media aspects conforming to (1), including a specific implementation of the M6d and M7d service APIs.

4. The MSE Description Document (2a) and a specific implementation of the abstract APIs (2b).

The MSE Description Document describes the features implemented by the MSE. The abstract APIs allow an external Android process to retrieve this document and configure the SDK with a set of configurable parameters that are described in the MSE Description Document. They also allow it to interrogate the state and status of the running SDK.

### 5.2.5 Benefits

The benefits of the above approach are the following:

1. The MSE specification defines all mandatory and optional features in a single document, the MDD, with references to the specific relevant clause(s).

2. The MSE specification also optionally defines the MSE Configuration APIs for managing and retrieving information from an implementation.

3. An implementer can use the MSE specification’s MDD as a feature checklist.

4. An implementer can use the MSE Configuration API to implement the API for MSE services.

5. The SDK instantiation of an MSE specification includes a side MDD describing the features supported by the SDK and the optional configurations it may have.

6. The MSE service instantiation of an MSE specification includes an MSE configuration API conforming to the one defined in the MSE specification that can be used for retrieving and configuring the service.

7. The MSE service instantiation provides an MDD (as a side or as part of retrieval through MSE configuration API) that provides the supported features of the MSE service instantiation.

## 5.3 MSE framework proposal #2

### 5.3.1 Overview

The proposal follows existing practices in 3GPP for device and network APIs, as introduced in clause 4.2, and extends the documentation with best practices identified in other organizations as introduced in clause 4.3.

### 5.3.2 Architecture

The basic concept of the Media Service Enabler is to support third-party media delivery over the 5G System. Figures 5.3.2-1 and 5.3.2-2 provide an overview for an application that is deployed on top of a 5G System. In this case, the Application Provider is operating an external DN and connects to the 5G System using N6 for data delivery and possibly N33 to use specific 5G network services. A UE-resident application makes use of device functions (for example hardware and software exposed through APIs) and connects to the Application Provider.



Figure 5.3.2-1: Third-party application on top of 5G System – 5G System Architecture



Figure 5.3.2-2: Third-party application on top of 5G System – Interfaces

Figure 5.3.2-3 now extends the above basic architecture to provide to the Application Provider a set of 3GPP-specified functions, possibly both on UE and network side, in order to simplify operations. These functions are bundled as a Media Service Enabler (MSE) and offered to the Application Provider as follows:

- The service may be provisioned on the network side using an MSE Application Function. The provisioning reference point is summarized as MSE-1.

- User plane data may be exchanged with the Application Provider using an Ingest/Egest interface, MSE-2. Generally, this is a generic IP-based interface that directly uses N6 and the UPF. However, the MSE may offer specific Application Server functions at MSE-2.

- On the UE side, the functions of an MSE Client are accessed through a well-defined client API, MSE-6, that is aligned with other device APIs. The MSE Client may make use of other device functions that are expected to be accessible via existing device APIs.

- The MSE Client may be decomposed into C*ore Functions* defined in the relevant Media Service Enabler specification, and *External Device Reference Functions* that are accessed through well-defined APIs MSE-7.

- The MSE Client connects to the 5G network and may make use of Application Functions associated with this Media Service Enabler. Those functions are exposed through MSE-5.

- User data is exchanged with the MSE Application Server (if any) through MSE-4, which may define specific requirements on the usage of protocols, codecs, formats etc.



Figure 5.3.2-2: Addition of MSE to 5G-based media delivery

Providing a Media Service Enabler in this form has several benefits:

- The Application Provider has a set of functions that can be easily accessed in the same way that device functions are accessed today, namely through well-defined device APIs. The Application Provider can also use regular IP connectivity to operate its application.

- For the MSE developer, the focus is on providing a well-defined set of functions that are exposed to the application through MSE-1 and MSE-2 on the network side, and via MSE-6 on the UE device side.

- The MSE developer may provide the MSE Application Function and Application Server as well as the MSE Client. In this case, the primary interoperability aspects are at reference points MSE-1 and MSE-6.

In another case, the network functions for MSE may be provided by a 5G System operator. In this case the MSE Client and MSE AF are expected to also implement the functions and interoperability defined at reference points MSE-4 and MSE-5.

### 5.3.3 Functions and reference points

The following functions are defined:

- *Application:* A downloadable or installed application in a UE that makes use of the MSE to provide a Media Service to a user.

- *MSE Client*: A UE-internal function dedicated to a specific Media Service Enabler. The MSE Client is a logical function and its subfunctions may be distributed within the UE according to implementation choice. For example, it may define new core functions as well as referencing existing functions that are required to complete the expected functions.

- *MSE Application Function*: An Application Function similar to that defined in clause 6.2.10 of TS 23.501 [2], dedicated to a specific Media Service Enabler.

- *MSE Application Server*: An Application Server dedicated to a specific Media Service Enabler.

The following reference points, interfaces and APIs are defined:

- *MSE-1 (MSE Provisioning API):* External API, exposed by the MSE AF, which enables the Application Provider to provision the usage of the MSE.

- *MSE-2: (MSE Ingest/Egest API):* Optional external API exposed to the Application Provider by the MSE AS and used when the MSE AS in the trusted DN is selected to process content for the MSE.

NOTE: MSE-3 may be used for communication between MSE AS and MSE AF, but is not considered relevant.

- *MSE-4: (MSE User Plane interface):* Interface used by an MSE Client to exchange user data with an MSE AS.

- *MSE-5: (MSE Control API):* APIs exposed by an MSE AF to the MSE Client to configure and control MSE functions.

- *MSE-6: (MSE Client APIs):* APIs exposed by the MSE to the Application for client-internal communication to make use of MSE functions

- *MSE-7: (External Device API):* APIs exposed by the UE device to the MSE to make use of resident client functions such as rendering, playback, etc.

- *MSE-8: (Application APIs):* Interface used for information exchange between the Application and the Application Provider.

### 5.3.4 Specification

Media Service Enabler specifications do not attempt to define an entire service, but only a subset of small defined functions. Hence, it is essential to understand that whatever is not defined to complete a service does not need to be documented. An MSE specification is a bottom-up specification: it specifies what is needed and does not address what is not needed.

An MSE specification is proposed to include the following information:

1. *Pre-requisites and Assumptions* (Highly recommended): Pre-requisites document what is expected to be available either from the 5G System (i.e. certain functionalities of the 5G System) or from implementation (for example functions available on the device). These pre-requisites may be considered to be part of the specification (as reference to an external specification), but it is important to identify this separately in order to clearly demarcate the boundaries of the MSE with respect to other functions. Example for pre-requisites include, but are not limited to:

a) Existing and required device functions and the corresponding APIs defined as MSE-7.

b) Existing and required 5G System functions.

2. *Overall specification of the function, including a specific architecture* (Highly recommended). This includes:

a) Instantiations of the MSE reference points and functions.

b) A typical call flow.

3. *Specification of the MSE Client functions and the corresponding MSE-6 APIs* (Highly recommended). This typically includes functionalities such as configuration, settings, notifications, events, data and status query as well as functional methods. It includes:

a) Definition of the internal functions itself.

b) Definition of how to use existing and required device functions.

c) Strict definition of the API methods with details such as name, pseudo code, functions. As a common language IDL or C is proposed to be used.

4. *Control Plane API and network/MSE Application Function* (Highly recommended)

a) Definition of the internal functions of the AF, using common practices of a RESTful API

b) Alignment with 5G Media Streaming functionalities as defined at reference point M5 of TS 26.501 [13] and TS 26.511 [6], using OpenAPI/YAML.

5. *User plane reference point and network/MSE Application Server* (Optional but recommended)

a) Definition of internal functions of the Application Server, based on common Internet protocols, preferably by reference to external specifications (IETF, MPEG, etc.)

### 5.3.5 Implementation support beyond specification

Beyond the specification, it is proposed to document guidelines and additional support material for developers. The following aspects are considered:

1. *Guidelines for application developers* (Highly Recommended)

- Providing guidance on how an application developer can make use of the Media Service Enabler.

- This is preferably done by providing examples and implementation hints.

2. *Guidelines for MSE implementers* (Optional):

- Providing guidance to an implementer of an MSE Client and/or AF in order to support implementation. Such guidelines may also be provided in line with the specification text.

- If provided, the guidelines are preferably separated in style and form from the main specification text. For example, this may be added in a specific "box" or "frame" that identifies this as an informal implementation hint.

3. *Considerations on device API implementations* (Recommended)

- The device APIs MSE-6 and MSE-7 are typically only documented on a conceptual level.

- Considerations on the specifics for implementing the APIs, for example in Android as RESTful APIs in devices, is relevant.

4. *Considerations of a Conformance Test Suite* (Optional, but expected to be at least considered):

- A Conformance Test Suite is a collection of tests covering the breadth of the MSE functions. The tests include the definition of test cases, the definition of test assets as well as the success criteria to pass the tests. A typical figure for a test application to test the implementation of the MSE Client is shown in Figure 5.3.5-1.

- The considerations documented are expected to allow third parties to implement a full Conformance Test Suite in order to test the 3GPP-defined APIs and conformance for correct implementation. Follow-up such as adopter programs may be considered.

- The Conformance Test Suites and adopter program may be provided by external organizations, for example 3GPP market representation partners (MRPs).



Figure 5.3.5-1: Test Framework for MSE Client Implementation

### 5.3.6 Style and documentation guidelines

The primary goal is to achieve consistency across the API, as well as across all specifications. Consistency makes it easier for developers, editors, reviewers, and users of the documentation to understand and modify it. While each organization and specification may and should have its own look and feel, it is considered appropriate to establish a style guide convention. The Style Guide of the OpenXR Documentation has been branched from the Vulkan documentation and is hence considered a broadly adopted and established convention. In addition, 3GPP uses OpenAPI for the API definition towards the network.

Hence, it is proposed to align with the style guide and documentation conventions from OpenXR as well as OpenAPI as follows:

1. Develop APIs for the relevant reference points in a Github- or gitlab-based environment and only port agreements or full specifications to 3GPP specifications. The development of the formal APIs is done in a git-based environment.

2. For device-internal API definitions, align with the OpenXR style guide https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html as follows:

- Use Asciidoc <http://www.asciidoctor.org/> to the extent possible to define formal APIs.

NOTE: References to the Asciidoctor User Manual are to sections in the document at <http://asciidoctor.org/docs/user-manual/>

- For API naming conventions, it is proposed that the rules defined in [https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html#naming](https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html" \l "naming) apply with the following adaptation:

- Each MSE is assigned a prefix (for example MSE). In similar way as XR is used in the OpenXR spec, an equivalent usage of MSE is expected for an MSE spec. ~~This prefix is used as XR is used in the description above~~.

- Prefixes are used in the API to denote specific semantic meaning of MSE names, or as a label to avoid name clashes, and are explained here:

* + MSE/Mse/mse
  + All types, commands, enumerates and C macro definitions in the specification are prefixed with these characters, according to the rules defined above.

- For the markup style, it is proposed that the ETSI/3GPP documentation rules as well as the rules defined in https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html#markup apply. In particular, section 5.7 on writing reference pages is expected to apply: [https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html#writing-refpages](https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html" \l "writing-refpages)

- Provide reference pages for the MSE according to the OpenXR principle https://registry.khronos.org/OpenXR/specs/1.0/man/html/openxr.html

3. For the network-based APIs and reference points, define RESTful APIs and use the conventional OpenAPI rules as defined by 3GPP in TS 29.501 [16].

4. For regular data communication reference to existing protocols and formats.

Editor’s Note: The W3C process is not yet documented and some aspects of the W3C process may also be considered https://www.w3.org/developers/tools/

### 5.3.7 Examples

#### 5.3.7.1 Example 1: MBMS Client

Based on the specification template in clause 5.3.5 and the style guidelines in clause 5.3.6, table 5.3.7-1 provides a potential mapping of the MBMS Client function, as introduced in clause 4.2.1, to the MSE concept.

Table 5.3.7.1-1 Mapping of MBMS Client to MSE concept

|  |  |  |
| --- | --- | --- |
| MSE Specification | Specification | Comments |
| Pre-requisites and assumptions | TS 26.347, clause 6.1: Background | Some high-level aspects are discussed, but a detailed listing of pre-requisites and assumptions is not available. |
| Overall specification of the function including a specific architecture | TS 26.347, clause 5: Reference Client Architecture | A reference architecture is provided, but no high-level call flows. |
| Specification of the MSE Client functions and the corresponding MSE-6 APIs | TS 26.347, clause 6.2, 6.3 and 6.4 for different APIs | Detailed specification of the APIs. However, improved documentation using well-defined types, structures, highlighting and so on could be applied. |
| Control Plane API and network/MSE Application Function | TS 26.346, clause 6 and 9 on user services and referenced in TS 26.347 | The control plane API is not defined explicitly, but as a general protocol. |
| User Plane reference point and network/MSE Application Server | TS 26.346, clause 7, 8, 8A, and 8B and referenced in TS 26.347 | The user plane reference point is defined explicitly, but as a general protocol. |
| Guidelines for application developers | TS 26.347, annex E | Some high-level implementation guidelines are provided. More detailed call flows would be needed. |
| Guidelines for MSE developers | TS 26.347, clause 6.2.2, 6.2.3 and 6.2.4, MBMS Client State Model | A detailed set of basic implementation ideas for the internal handling of an MBMS Client is provided as part of the description. |
| Considerations on device API implementations | TS 26.347, Annex B, Interface Definition Language for MBMS-APIs | A full IDL-based interface definition is provided, but it is informative. It is also not provided as “code”, but as text in the document. |
| Considerations for a Conformance Test Suite | (Not existing) | Nothing is documented on this matter. However, as seen in clause 4.2.1, Android APIs exist |
| Style and documentation | TS 26.347, annex A | Style and documentation is weak. Annex A introduces the usage of IDL, but is lacking compared to clause 5.3.6:  - No git based approach  - No usage of ASCIIDOC  - No consistent API naming conventions are applied  - No markup or reference pages are generated  - No OpenAPI-based network protocols are defined. |

In summary, the MBMS Client, as currently specified by 3GPP, quite closely follows the definition of an MSE. Because the MBMS Client and the APIs were developed in stages, the documentation is not consistent in one specification, but rather is spread over several documents. However, most of the considered information is present. An improved overall documentation process, more style guidelines and so on would be needed.

#### 5.3.7.2 Example 2: DASH Player

Based on the specification template in clause 5.3.5 and the style guidelines in clause 5.3.6, table 5.3.7.2-1 provides a potential mapping of the DASH Player function, as introduced in clause 4.2.2, to the MSE concept.

Table 5.3.7.2-1 Mapping of DASH Player to MSE concept

|  |  |  |
| --- | --- | --- |
| MSE Specification | Specification | Comments |
| Pre-requisites and assumptions | TS 26.512, clause 13.2. The Media Playback and Content Decryption Platform external APIs. | Reference to TS 26.511 which in itself expects availability of playback based on CMAF playback requirements. |
| Overall specification of the function including a specific architecture | TS 26.501, clause 5.4, DASH Streaming and TS 26.512, clause 13.2. | Provides call flows and architecture. |
| Specification of the MSE Client functions and the corresponding MSE-6 APIs | Corresponds to the M7d APIs as defined TS 26.501 and TS 26.512, clause 13.2. | Definition of internal functions, methods, notification and error events, and status information in a formal manner. Reference to dash.js documentation. |
| Control Plane API and network/MSE Application Function | Not defined (unless MPD is considered control plane) |  |
| User plane reference point and network/MSE Application Server | Defined as M4d in TS 26.501 and TS 26.512, clause 10. | References TS 26.247 and ISO/IEC 23009-1. |
| Guidelines for application developers | Examples are provided by reference to dash.js |  |
| Guidelines for MSE developers | Partially provided in TS26.247 and reference to ISO/IEC 23009-1, Annex A |  |
| Considerations on device API implementations | Nothing available |  |
| Considerations of a Conformance Test Suite | Nothing referenced | DASH-IF defines reference player. |
| Style and documentation | TS 26.347, annex A | Style and documentation is weak. Annex A introduces the usage of IDL, but is lacking compared to clause 5.3.6:  - No git based approach  - No usage of ASCIIDOC  - No consistent API naming conventions are applied  - No markup or reference pages are generated  - No OpenAPI-based network protocols are defined. |

## 5.4 Discussion on different MSE framework proposals

Two different approaches for an MSE framework are provided in clause 5. The approaches share many similarities, in particular:

* Defining the key concepts of MSE
* Functional definitions of the Media Service Enabler.
* Media Service enabler is a set of mandatory and possibly optional set of functionalities
* Definition of device-internal APIs and network interfaces.
* Support of specification and implementations
* Easily mapped to SDK implementation

However, there are also complementary aspects:

* Approach 1, as proposed in clause 5.2, addresses the following additional aspects
  + A document for cataloging a MSE specification’s features and their options.
  + Configuration of the Media Service Enabler by supplying configuration parameters as needed by the user of the MSE
  + capability discovery within the Media Service Enabler. This may include aspects that are binary (supported, not supported), but could also be more nuanced, and/or optionally cataloguing the subset of the specification features supported by an implementation in a document and their implemented options.
* Approach 2, as proposed in clause 5.3, addresses the following additional aspects
  + Reference architecture for MSE based on 5GMS architecture
  + Template for Media Service Enabler specification drafting
  + Addressing aspects beyond specification, namely test, reference implementations, as well as conformance considerations
  + Tooling, style and documentation guidelines

# 6 MSE Specification Framework

## 6.1 Introduction

This clause provides an MSE Specification framework based on the examples and framework considerations in clause 4 and 5, respectively.

Based on the analysis in clause 5.4, it is considered that the approach in clause 5.3 is used as the baseline for the MSE specification initial framework, but the concepts for 5.3 are beneficially enhanced adding the complementary concepts of the approach in 5.2. Possible extensions of the framework to provide more consistent deployments can be considered later.

## 6.2 Initial MSE framework

### 6.2.1 General Concepts

The basic concept of the Media Service Enabler is to support third-party applications to make use of advanced functionalities provided by the 5G System, combined with additional well-defined client and network functionalities for media services: an MSE enables improved media services.

In implementations and deployments, such packaged functions are typically referred to as a Software Development Kit (SDK) and they are usable by applications through well-defined APIs. A few potential properties of a Media Service Enabler are provided:

* A set of functions that may be used to deploy applications that can make simple use of 5G System functionalities.
* A set of robust features and functionalities which reduce the complexity of developing applications.
* Functions to leverage system and radio optimizations as well as features defined in 5G System (5G Core Network and 5G NR).
* Usability of the set of functions by well-defined and well-documented device APIs.
* Provision of network interfaces to connect to the 5G System.
* A testable set of functions. Testing and conformance may be addressed outside 3GPP, for example by a Market Representation Partner (MRP) such as 5G-MAG or by an industry forum.
* Guidelines and examples to make use of the set of functionalities provided by an MSE.

A general initial idea on how to define Media Service Enablers is documented below:

* Combine functions defined in 3GPP (for example a codec) and/or reference technologies defined outside 3GPP, for example in MPEG or Khronos, and provide relevant subsets and profiles of these.
* Include mandatory, recommended and optional functions.
* Define signaling and capability negotiation for all functions.
* Specify requirements for client and network functions, as needed.
* Include relevant functions such as QoE metrics and KPIs.

Providing a Media Service Enabler in this form has several benefits:

- The Application Provider has a set of functions that can be easily accessed in the same way that device functions are accessed today, namely through well-defined device APIs. The Application Provider can also use regular IP connectivity to operate its application.

- For the MSE developer, the focus is on providing a well-defined set of functions that are exposed to the application through MSE-1 and MSE-2 on the network side, and via MSE-6 on the UE device side.

- The MSE developer may provide the MSE Application Function and Application Server as well as the MSE Client. In this case, the primary interoperability aspects are at reference points MSE-1 and MSE-6.

In another case, the network functions for MSE may be provided by a 5G System operator. In this case the MSE Client and MSE AF are expected to also implement the functions and interoperability defined at reference points MSE-4 and MSE-5.

In the remainder of this clause, an MSE reference architecture is provided and functions and interfaces are defined.

### 6.2.2 MSE Reference Architecture

The basic concept of the Media Service Enabler is to support third-party delivery of media over the 5G System. Figure 6.2.2-1 provides the Application Provider with a set of 3GPP-specified functions, possibly both on UE and network side, in order to simplify operations. These functions are bundled as a Media Service Enabler (MSE) and offered to the Application Provider as follows:

- The service may be provisioned on the network side using an MSE Application Function. The provisioning reference point is summarized as MSE-1.

- User plane data may be exchanged with the Application Provider using an Ingest/Egest interface, MSE-2. Generally, this is a generic IP-based interface that directly uses N6 and the UPF. However, the MSE may offer specific Application Server functions at MSE-2.

- On the UE side, the functions of an MSE Client are accessed through a well-defined client API, MSE-6, that is aligned with other device APIs. The MSE Client may make use of other device functions that are expected to be accessible via existing device APIs.

- The MSE Client may be decomposed into C*ore Functions* defined in the relevant Media Service Enabler specification, and *External Device Reference Functions* that are accessed through well-defined APIs MSE-7.

- The MSE Client connects to the 5G network and may make use of Application Functions associated with this Media Service Enabler. Those functions are exposed through MSE-5.

- User data is exchanged with the MSE Application Server (if any) through MSE-4, which may define specific requirements on the usage of protocols, codecs, formats etc.



Figure 6.2.2-1: Media Service Enablers in 5G Systems

### 6.2.3 Functions and reference points

The following functions are defined:

- *Application:* A UE-resident function that uses the Media Service Enabler to create a service or a user experience

- *MSE Client*: A UE-internal function dedicated to a specific Media Service Enabler. The MSE Client is a logical function and its subfunctions may be distributed within the UE according to implementation choice. For example, it may define new core functions as well as referencing existing functions that are required to complete the expected functions.

- *MSE Application Function*: An Application Function similar to that defined in clause 6.2.10 of TS 23.501 [2], dedicated to a specific Media Service Enabler.

- *MSE Application Server*: An Application Server dedicated to a specific Media Service Enabler.

The following reference points, interfaces and APIs are defined:

- *MSE-1 (MSE Provisioning API):* External API, exposed by the MSE AF, which enables the Application Provider to provision the usage of the MSE.

- *MSE-2: (MSE Ingest/Egest API):* Optional external API exposed to the Application Provider by the MSE AS and used when the MSE AS in the trusted DN is selected to process content for the MSE.

NOTE: MSE-3 may be used for communication between MSE AS and MSE AF, but is not considered relevant.

- *MSE-4: (MSE User Plane interface):* Interface used by an MSE Client to exchange user data with an MSE AS.

- *MSE-5: (MSE Control API):* APIs exposed by an MSE AF to the MSE Client to configure and control MSE functions.

- *MSE-6: (MSE Client APIs):* APIs exposed by the MSE to the Application for client-internal communication to make use of MSE functions

- *MSE-7: (External Device API):* APIs exposed by the UE device to the MSE to make use of resident client functions such as rendering, playback, etc.

- *MSE-8: (Application APIs):* Interface used for information exchange between the Application and the Application Provider.

### 6.3 MSE specification Template

The following is a template for a Media Service Enabler specification.

1 Scope

2 References

3 Terms and abbreviations

4 General

4.1 Overview

- Motivation and scope for the MSE.

- Applied MSE principles.

- Overview of specification.

- Addressed functionalities.

4.2 Typical Use Cases and applications

- Use Cases that may be addressed with the MSE.

- Applications benefitting from the MSE.

5 Reference architecture and procedures

5.1 Reference Architecture

- Instantiation of the general MSE architecture.

- Providing the defined APIs and reference points.

5.2 Core functions and extensions

- Core functions are all functions that need to be implemented to support the MSE (required).

- Extensions define a set of functions that are only required for certain use cases (optional).

- Configuration parameters for the core functions and extensions

- Capability options for the core functions and in particular for the extensions

5.3 Procedures and call flows

- High-level call flows and procedures for the most common use cases.

- Summary of specified reference points and procedures in this specification pa.

6 Prerequisites

6.1 5G System functionalities

- Functionalities required of the 5G System with reference.

- May address requirements dependent on core functions or as part of an extension only.

6.2 Device APIs and functionalities

- Requirements and functionalities needed from the device.

- Definition of reference point MSE-7 requirements, i.e. reference APIs and functionalities.

7 MSE Application Function specification

7.1 Overview of the MSE API calls

7.2 Functionality

- Defines states of the MSE AF in relation to the MSE client. Examples for state are IDLE, REGISTERED, ACTIVE, etc.

7.3 MSE AF API methods and parameters

- RESTful APIs.

- MSE-5 is defined.

- MSE-1 may be defined.

8 MSE user plane specification

8.1 Overview

- Definition of MSE-4 interface.

- Protocol Stack.

8.2 User plane configuration protocols

8.4 Content delivery protocols

8.3 Formats and Codecs

8.4 QoS considerations

8.5 Security considerations

9 MSE Client specification

9.1 Overview of the MSE API Calls

9.2 Functional description

- Uses reference pre-requisites, user plane functionality, control plane, and client API.

- Defines states of the MSE client in relation to the application. Examples for state are IDLE, REGISTERED, ACTIVE, etc. State changes may occur through or by information received through MSE-6 the network interface.

- A set of client-internal reference parameters that are changed based on either configuration or API calls through MSE-6 or by information received through the network interface MSE-4 or MSE-5.

- Metrics, data and KPI collections, for example to be provided to analytics servers.

9.3 MSE Client API methods and parameters

- Different methods that allow the application to communicate with the MSE client. For each method, the following information is provided:

- A high-level description of the method.

- An example call flow.

- The parameters that are exchanged as part of the API call.

- The usage of the API by the application.

- The MSE Client actions, including pre and post conditions.

- Configuration

- Capabilities

This API typically includes functionalities such as configurations, settings, notifications, events, data and status query as well as functional methods. As an example, the API may provide the ability to query metrics and KPIs,

- Optional capability discovery: discovery of the capabilities supported by an implementation including the additional configuration parameters specific to that MSE implementation

- Specification using a well-defined language, for example C or IDL,

Annex A (informative): Implementation guidelines

A.1 Guidelines for application developers

- Use Case mapping.

The guidelines are expected to provide guidance how an application developer can make use of the MSE. This is preferably done by providing examples and implementation hints.

A.2 Guidelines for MSE implementers and reference implementations

- The guidelines are expected to provide guidance to an MSE Client and/or AF implementor in order to support implementation.

- A reference implementation of the MSE may be considered.

Annex B (informative): Considerations on conformance testing

B.1 Overview

- A Conformance Test Suite is a collection of tests covering the breadth of the MSE functions. The tests include the definition of test cases, the definition of test assets as well as the success criteria to complete the tests.

B.2 Potential testing framework

B.3 Potential test cases

B.4 Potential conformance testing procedures

Annex C (normative): API Reference Pages

Annex D (informative): Considerations on API Instantiations

- The device API implementations MSE-6 and MSE-7 are typically only done on a conceptual level.

- Considerations on specifics for the instantiations of the APIs, for example in Android or in web browsers.

Annex E (informative): Attachments and online repositories

## 6.4 Beyond the MSE Specification – guidelines, tests and reference implementations

Beyond the MSE specification, and as indicated in clause 6.3, the following aspects are considered in the annexes for the specification template:

* Guidelines for application developers.
* Guidelines for MSE implementers and reference implementations.
* Device API instantiations.
* Conformance Test Suite.

Such efforts are not necessarily suitable for 3GPP working processes. Hence, collaboration with other organizations, such 3GPP market representation partners (MRPs) or open-source projects may be considered. The annexes indicated above may initially contain only considerations that can be used by third parties in order to develop their own implementations, guidelines, test frameworks and reference implementations.

As an example, the development of a reference implementation of MSE Client and network functions can support developers and Application Providers to quickly gain access to newly defined functionalities. This is, for example, shown in figure 6.4-1 for which reference implementations of the MSE are used as part of a reference, demonstration or production application. In this case, the reference implementation makes use of existing device functions and 5G System functions. As an example, the 5G-MAG reference tools <https://www.5g-mag.com/reference-tools> provide an approach to developing such reference implementations.



Figure 6.4-1: MSE Reference Implementation

As another example to support the specification development, a conformance test suite may be developed in order to test the 3GPP-defined APIs and conformance for correct implementation. A framework for this is provided in figure 6.4-2.



Figure 6.4-2: Test/Conformance Framework for MSE Client Implementation

In this case, a test framework is developed in order to test the functionality of the MSE Client implementation. If all tests are passed, the MSE Client may be considered conformant to the specification. Such an approach may be even extended to create an adopter program, i.e. providing a process that allows an MSE implementation to officially claim support of the MSE specification by having verified that the all tests have been passed.

While 3GPP is not in a position to mandate such a conformance regime, it is highly recommended to consider the potential benefits of supporting third parties in developing suitable test and conformance programs.

# 7 Writing MSE Specifications: Style Guides and Tools

Editor’s Note:

What are reasonable abstraction languages?

* YAML/OpenAPIs => RESTFul APIs
* IDL
* others

Can we help documentation and specification using more tools?

* MSC
* ASCIIDOC
* IDL Editors
* Editing in markdown?

Hence, it is proposed to align with the style guide and documentation conventions from OpenXR as well as OpenAPI as follows:

* Use a github or gitlab based development of APIs and reference points and only port agreements or full specs to 3GPP specifications. The development of the formal APIs are done in a git-based environment.
* For the device-internal API definition, align with the OpenXR style guide https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html as follows:
  + Use Asciidoc <http://www.asciidoctor.org/> to the extent possible to define formal APIs. References to the Asciidoctor User Manual are to sections in the document at <http://asciidoctor.org/docs/user-manual/>
* For API naming conventions, it is proposed that the rules defined in [https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html#naming](https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html" \l "naming) apply with the following adaptation:
  + Each MSE gets assigned a prefix (for example MSE). This prefix is used as XR is used in the description above.
  + Prefixes are used in the API to denote specific semantic meaning of MSE names, or as a label to avoid name clashes, and are explained here:
  + MSE/Mse/mse
  + All types, commands, enumerates and C macro definitions in the specification are prefixed with these characters, according to the rules defined above.
* For the markup style, it is proposed that the ETSI/3GPP documentation rules as well as the rules defined in https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html#markup apply. In particular clause 5.7 on writing reference pages is expected to apply: [https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html#writing-refpages](https://registry.khronos.org/OpenXR/specs/1.0/styleguide.html" \l "writing-refpages)
  + Provide reference pages for the MSE according to the OpenXR principle https://registry.khronos.org/OpenXR/specs/1.0/man/html/openxr.html
* For the network-based APIs and reference points, define RESTful APIs and use the conventional OpenAPI rules as defined by 3GPP in TS 29.501 [29.501].
* For regular data communication reference to existing protocols and formats.
* Reference to Annex A
* We will also check if any of the W3C methods can be applied as well
  + <https://www.w3.org/developers/tools/>
  + https://github.com/w3c/respec

For SA4#121-e, a workflow and tools will be prepared.

# 8 Potentially Relevant 5G Media Service Enablers

Editor’s Note:

- collect MSEs that may be defined

- identify existing 3GPP specifications that may be improved using MSE concept

- MBMS/MBS Client,

- 5GMS Media Player

- identify new enablers that may fit into the MSE concept

- Split Rendering

- AI/ML functionalities

# 9 Conclusions and Recommendations

* summarize the document
* recommend the use of what is provided in clause 6, 7, and 8.

Annex A (informative): Details on Tools and Templates

* Provide details information on the tools and the work flow templates
* Provide install instructions
* Examples: ASCII-Doc, MSC, github repositories
* Online repositories
* Atttachments

Annex <X> (informative):  
Change history

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change history** | | | | | | | |
| **Date** | **Meeting** | **TDoc** | **CR** | **Rev** | **Cat** | **Subject/Comment** | **New version** |
| 2022-02 | SA4#117 | S4-220031 |  |  |  | Initial version | V0.0.1 |
| 2022-02 | SA4#117 | S4-220282 |  |  |  | Version agreed during SA4#117e | V0.1.0 |
| 2022-04 | SA4#118 | S4-220566 |  |  |  | Version agreed during SA4#118e | V0.2.0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2022-05 | SA4#119 | S4-220880 |  |  |  | Version agreed during SA4#119e | V0.3.0 |