ETSI TS 126 565 V18.0.0 (2024-07)

5G;
Split Rendering Media Service Enabler
(3GPP TS 26.565 version 18.0.0 Release 18)

Reference
DTS/TSGS-0426565vi00

Keywords
5G

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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Legal Notice

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. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found under https://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.

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

This specification defines a media service enabler for split rendering in the 5G system.

1 Scope

The present document defines a Media Service Enabler for Split Rendering according to the guidelines of TR26.857 [1]. The Split Rendering MSE covers functionality on the UE and on the Media AS. It also defines an API that is exposed to application developers on the UE to start and manage split rendering sessions.

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 26.857: "5G Media Service Enablers".

[2] ISO/IEC 12113:2022, Information technology, Runtime 3D asset delivery format, Khronos glTF 2.0

[3] ISO/IEC 23090-14: Information technology — Coded representation of immersive media — Part 14: Scene Description for MPEG Media.

[4] 3GPP TS 26.119, Media Capabilities for Augmented Reality

[5] 3GPP TS 26.506, 5G Real-time Media Communication Architecture (Stage 2)

[6] 3GPP TS 26.113, Real-Time Media Communication; Protocols and APIs

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

[8] 3GPP TS 26.522, 5G Real-time Media Transport Protocol Configurations

[9] 3GPP TS 26.510, Media Delivery: interactions and APIs for provisioning and media session handling

[10] Khronos, The OpenXR API, https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html

[11] W3C, WebXR Device API, WebXR Device API (immersive-web.github.io)

[12] Khronos, WebGL Specification 1.0, WebGL Specification (khronos.org)

[13] W3C, Web Audio API, Web Audio API (w3.org)

[14] 3GPP TS23.501, System architecture for the 5G System (5GS).

[15] 3GPP TS23.503, 5G; Policy and charging control framework for the 5G System (5GS).

[16] 3GPP TS26.857, 5G Media Service Enablers.

[17] 3GPP TS 26.247: "Transparent end-to-end Packet-switched Streaming Services (PSS); Progressive Download and Dynamic Adaptive Streaming over HTTP (3GP-DASH)".

3 Definitions of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms 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].

example: text used to clarify abstract rules by applying them literally.

3.2 Symbols

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

<symbol> <Explanation>

3.3 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].

AF	Application Function
AS	Application Server
MAF	Media Access Function
MAP	Media Application Provider
MSH	Media Session Handler
MSE	Media Service Enabler
5G-RTC	5G Real-Time Communication
RTC	Real-Time Communication
SR	Split Rendering
SRC	Split Rendering Client
SRS	Split Rendering Server
SWAP	Simple WebRTC Application Protocol
UE	User Equipment
XR	eXtended Reality

4 General

4.1 Overview

The Split Rendering Media Service Enabler collects a set of 5G media functions to build a media service enabler that targets application developers, network operators, and Media Application Providers, to enable the realization of split rendered applications.

The interfaces, formats, protocols, and APIs are either referenced or defined in this specification. This will allow for interoperability between multiple vendor implementations.

This specification targets primarily XR applications. However, it is not limited to XR applications and may be used for rendering for 2D displays.

4.2 Typical Use Cases

A typical use case for the split rendering MSE is immersive gaming. In this use case, the UE benefits from invoking split rendering by avoiding the download of the game to the phone and getting high quality graphics from edge rendering.

Another use case that can benefit from split rendering is immersive communication, where users gather in a shared space and interact with each other and with the environment. Users may be represented by sophisticated Avatars and as the number of users increases the rendering will become more complex.

5 Reference Architecture and Procedures

5.1 Reference Architecture

5.1.1 Introduction

In this clause, different variants of the reference architecture for the split rendering Media Service Enabler (MSE) are defined, each representing a different perspective and level of details.

The following functions are introduced:

- Split-Rendering Client (SRC): This function is responsible for discovering the UE media capabilities and negotiating with the Split-Rendering Server (SRS) to agree on the split-rendering process.
- Split-Rendering Server (SRS): This function is responsible for negotiation of SR session with SRC, monitoring the server’s edge resource usage, and managing/running the split rendering process.
- Media Application Function (AF): responsible for provisioning, QoS allocation, and edge resource discovery.
- Media Application Provider: The Media Application Provider that offers the service.
- Application: The application running on UE.
- Media Session Handler (MSH): is the entity on UE that is responsible for the control plane communication with the AF.
- XR Runtime: Set of functions provided by XR Device to the XR Application to create XR experiences.

5.1.2 Client Architecture

The client architectural breakdown is based on the client architecture in TS 26.119 [4] clause 5.1. The figure depicting the client architecture is replicated here as Figure 5.1.2-1 for convenience.

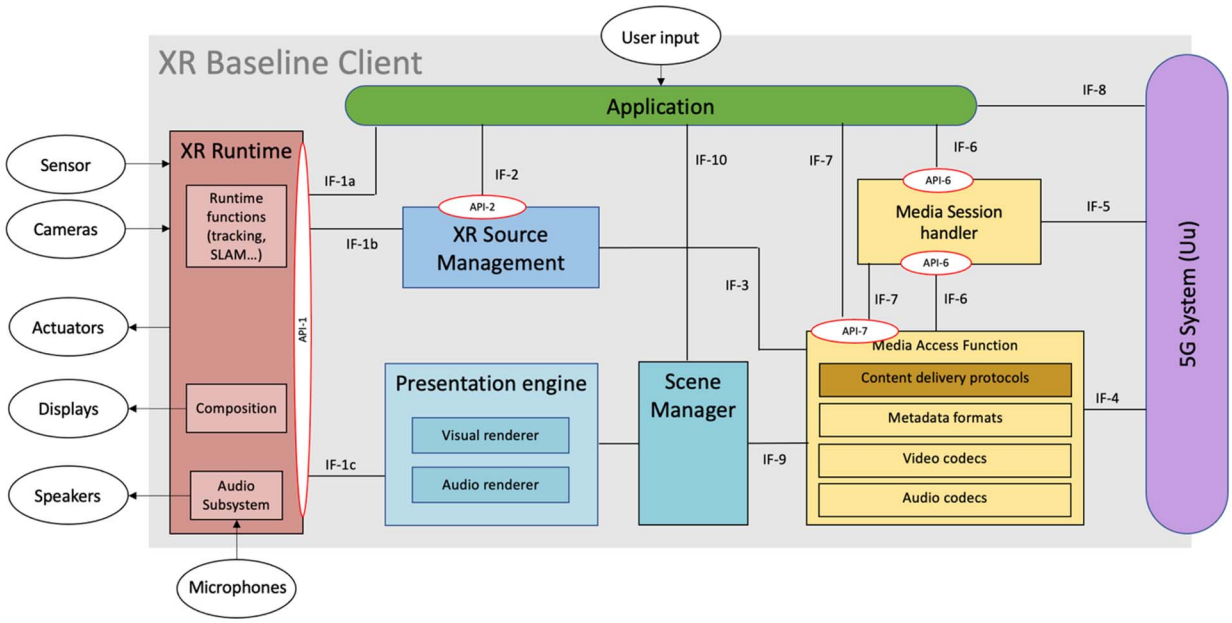

Figure 5.1.2-1 - XR Baseline terminal architecture

The split rendering client consists of the following components:

- The Media Access Functions: allows for fetching and processing of the pre-rendered media in preparation of final display. The MAF is also responsible for the carriage of any metadata or local media to the split rendering server.
- The scene manager and “thin” Presentation Engine: is responsible for the negotiation of the split rendering session and the parsing of the description of the rendered media as provided by the SRS. It is also responsible for setting up and managing the XR session with the XR runtime.
- The XR source management is responsible for gathering timed metadata such as pose and action information and sending it to the SRS.
- XR Runtime: Set of functions provided by XR Device to the XR Application to create XR experiences.

5.1.3 End-to-End Architecture

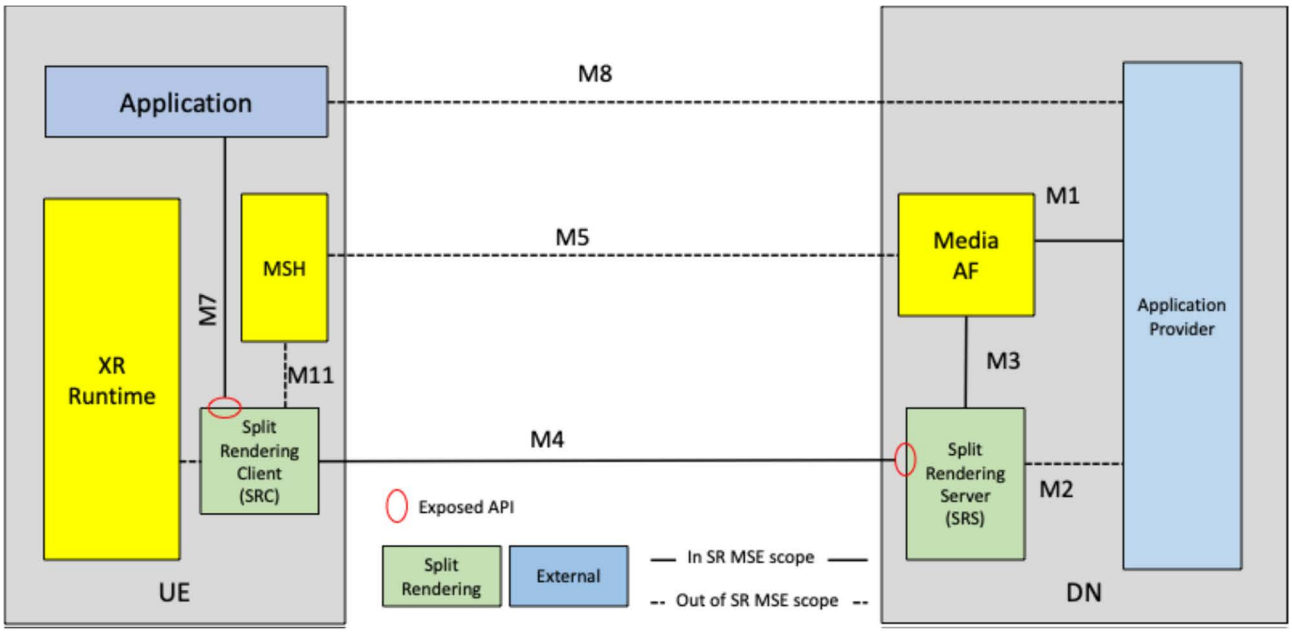

Figure 5.1.3-1 – Split management architecture

As shown in Figure 5.1.3-1:

1. The Media Application Providers (AP) provisions the split-rendering through M1.

2. In the use cases in which the MAP is involved in the media delivery, the M2 interface is used for this purpose.

3.The communication between Media AF and SRS is through M3. This interface is out of the scope of this document. This interface may for instance include the EDGE-3 interface.

4. The signaling as well as the media delivery between SRC and SRS is though M4.

5. The Media AF may provide SR-related information to the Media Session Handler (MSH) through the M5 interface, as defined in TS26.510.

6. The SRC in the UE discovers the application through M11 and handles the XR runtime.

7. The SRC discovers the client media capabilities through the M7 interface. This interface is out of the scope of this document.

8. The 5G Application and MAP interact through M8. This interface is out of the scope of this document.

5.1.4 User Plane Architecture

Figure 5.1.4-1 depicts the user plane architecture for split rendering.

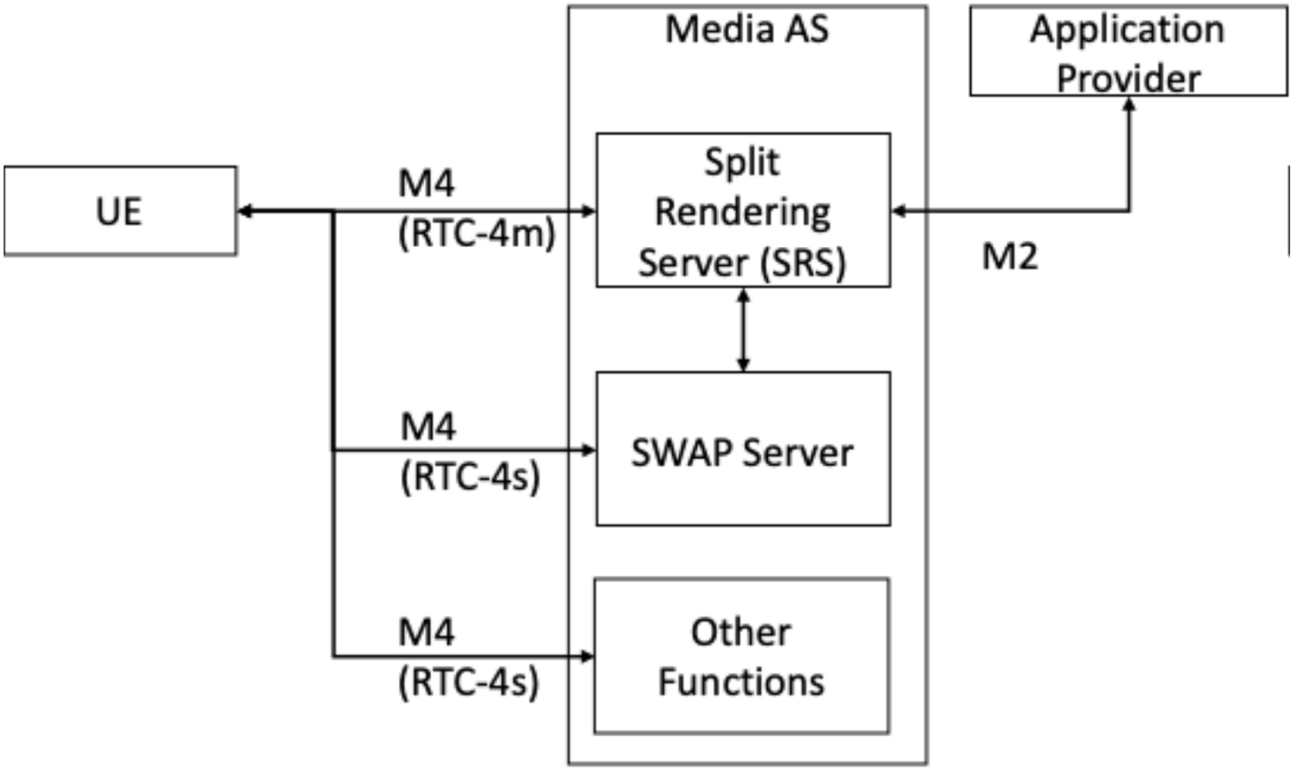

Figure 5.1.4-1 – User Plane Architecture for Split management architecture

In the context of split rendering, the M4 interface is further classified as RTC-4s and RTC-4m sub-interfaces. The RTC-4s interface covers all user-plane signaling, including WebRTC and ICE signaling. The RTC-4m serves for media and metadata exchange between the split rendering client and the split rendering server.

5.2 Procedures and Call Flows

5.2.1 Call flow for Split Rendering instance discovery

5.2.1.1 Call flow for edge server and split rendering session setup

Figure 5.2.1.1-1 demonstrates a general call flow for split-rendering.

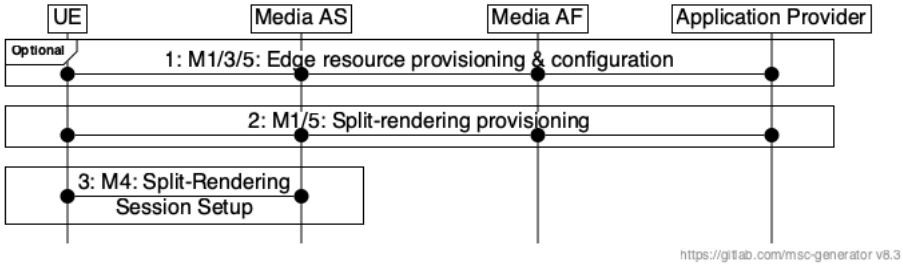

Figure 5.2.1-1: High-level call flow for split-rendering

Steps:

1. In this optional step, the Media Application Provider requests and sets up the edge server(s) used for the split-rendering as described in TS 26.506 clauses 6.1 or 6.2. The Media Application Provider may use any other method to allocation edge servers, or leave it to the MNO to set up appropriate edge servers to run the split-rendering process.

2. The Media Application Provider provisions the split-rendering session using M1 and M3, as defined in call flow of clauses 5.2.1.1. If the edge servers were provisioned in step 1, the edge servers ids are provided in this session to employ them for split-rendering.

NOTE: In the case of the client-driven edge management (TS 26.501 8.1), only the client-driven split-rendering (5.2.1.1) is applicable.

3. The split-rendering session is set up according to clause 5.2.2.

5.2.1.2 Client-driven procedures and call flows

Figure 5.2.1.2-1 demonstrates a call flow for setting up the split rendering by the client.

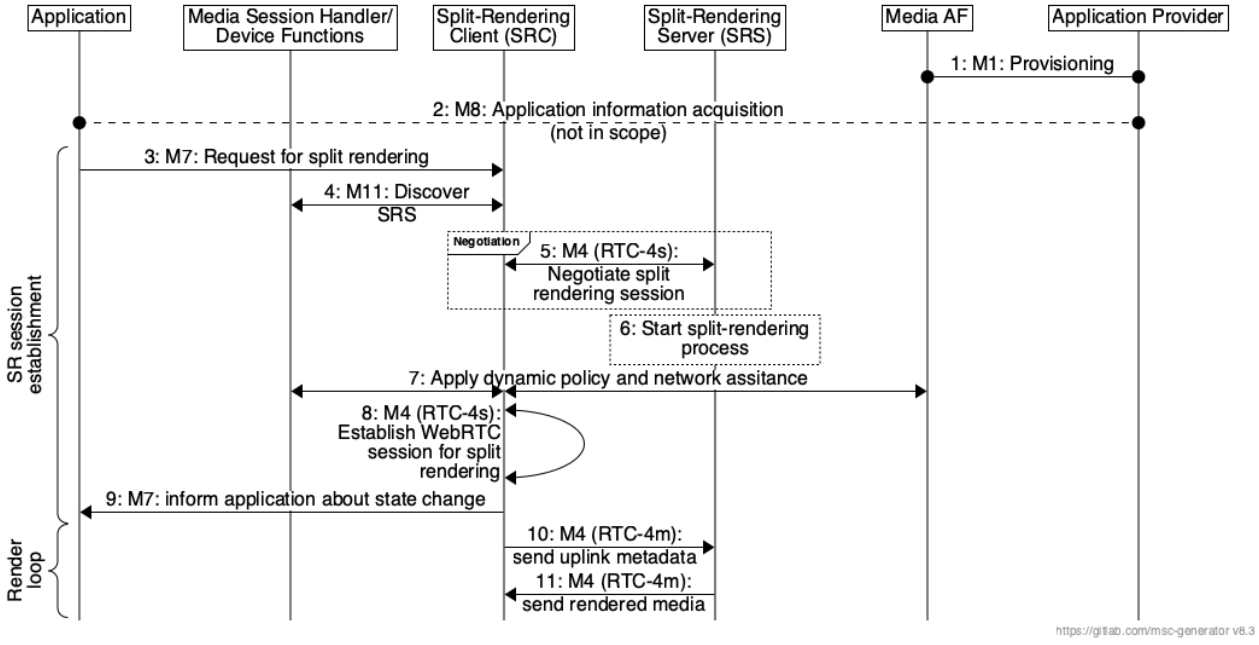

Figure 5.2.1.2-1: High-level call flow for initiating a split

Steps:

1. The Media Application Provider requests from the Media AF, the creation of a Provisioning session for split rendering.

2. The SR configuration is announced to the MSH as part of the Service Access Information.

3. The Application requests the deployment of split rendering from the SRC.

4. The SRC requests the discovery of a suitable SRS from the MSH. It may provide the client's media capabilities as input parameters.

5. The SRC and SRS negotiate the configuration of the split rendering session.

6. The SRS starts the split rendering process.

7. The SRC provides the session information via the M11 interface and requests the application of dynamic policy and subscription to network assistance from the Media AF, via the MSH.

8. The SRC establishes the WebRTC session.

9. The SRC informs the application that the split-rendering on edge is running via M7.

10. The SRC sends uplink metadata, such as pose and action information.

11. 0The SRS sends the rendered media to the SRC.

5.2.2 Call flow for Split Rendering session setup

The split rendering operation can be described by the call flow in Figure 5.2.2-1.

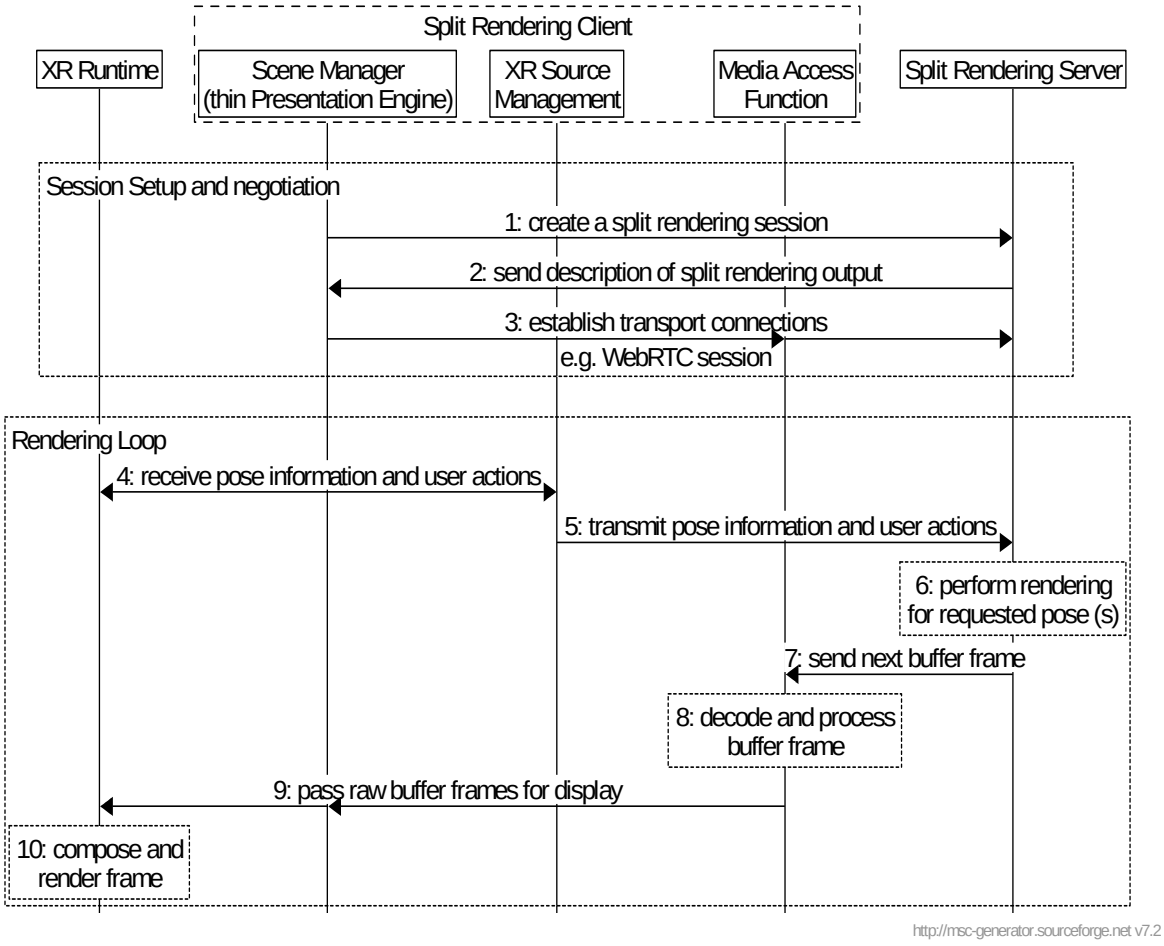

Figure 5.2.2- 1 High-level call flow for split rendering session setup and operation

The steps are:

1. The Presentation Engine discovers the split rendering server and sets up a connection to it. It provides information about its rendering capabilities and the XR runtime configuration, e.g the OpenXR configuration may be used for this purpose.

2. In response, the split rendering server creates a description of the split rendering output and the input it expects to receive from the UE.

3. The Presentation Engine requests the buffer streams from the MAF, which in turn establishes a connection to the split rendering server to stream pose and retrieve split rendering buffers.

4. The Source Manager retrieves pose and user input from the XR runtime.
5. The Source Manager shares the pose predictions and user input actions with the split rendering server.
6. The split rendering server uses that information to render the frame.
7. The rendered frame is encoded and streamed down to the MAF.
8. The MAF decodes and processes the received frame.
9. The MAF passes the decoded frame to the Scene Manager which passes it to the XR Runtime.
10. The XR runtime composes and renders the frame onto the display.

6 Prerequisites

6.1 General

The below section provides guidance on the pre-requisites on the 5G system and device APIs in order to host and run a split rendering session.

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 them separately in order to clearly demarcate the boundaries of the split rendering MSE with respect to other functions.

6.2 Pre-requisites on 5G System

Pre-requisites on the 5G System include, but are not limited to:

- The split rendering session is expected to operate in the 5G system as specified in [14] that supports functionalities and procedures to configure the quality of service (QoS) and charging information and functionality, which may be used by split rendering sessions.
- The split rendering session is expected to operate within the 5G system as specified in [14], which supports functionalities and procedures to configure policy and charging control (PCC) information as specified in [15]. Split rendering sessions may benefit from these procedures.
- The split rendering server functionality may operate within the 5G system as specified in |14], which supports the discovery of and access to edge resources. The edge functionality may be used to deploy split rendering server functionality.

6.3 Pre-requisites on Device APIs and Functionality

The following assumptions for the split rendering client are made:

- The SRC may have access to an XR runtime through a well-defined API such as the OpenXR [10] or WebXR [11] APIs.
- The SRC has access to 3D graphics library, such as. WebGL [12], and to an audio rendering engine, such as WebAudio [13].

7 Network Support

7.1 Overview

The Split Rendering MSE stands to benefit from the several procedures that the network offers. These include but are not limited to:

- Dynamic QoS and charging,
- Edge resources,
- Consumption and metrics reporting,
- Configuration Information.

In this clause, the control plane procedures that are relevant for split rendering and their usage are described.

7.2 Provisioning

A Media Application Provider that wishes to offer applications using split rendering shall use the procedures and data models defined in TS 26.510 [9] clauses 6 and 8 to create a Provisioning Session with the Media AF.

The ProvisioningSessionType shall be set to "`BIDIRECTIONAL`".

The aspId shall be configured and shall be a unique identifier for the Media Application Provider that offers split rendering.

The externalApplicationId shall be a URN that uniquely identifies the application and shall be terminated by the sub-string "+3gpp-sr". An examples is as follows: "urn:com:example:game+3gpp-sr".

7.3 Dynamic Policy and Network Assistance

Dynamic policy and network assistance may be provisioned by the Media Application Provider with the Media AF. The allowed dynamic policies for the split rendering sessions of the Media Application Provider are communicated to the MSH in the UE using the Configuration procedure.

Upon the creation of a new split rendering session and upon eligibility, the MSH shall use the Dynamic Policy API to request the allocation of network resources and charging policy to the session based on the information in the corresponding Provisioning session.

A policy template that is provisioned for split rendering should be associated with the split rendering configuration.

The MSH and the WebRTC Signaling Server shall support the dynamic policy API as defined in clause 10.4 of TS 26.113 [6].

The Media Application Provider may provision support for PDU Set marking. The SRS shall support the PDU Set marking and should support the End of Burst marking for the RTP streams that are generated by the Split Rendering Server.

7.4 Edge Resources

A Media Application Provider may use the procedures defined in TS 26.510 [9] clause 8.6 to define an edge resource configuration to be used for split-rendering sessions.

In this case:

- The eligibilityCriteria shall be present and shall have appRequest set to true.
- The easRequirements shall indicate "SR" as the easType and shall include "3gpp-sr" among the easFeatures. The serviceKpi shall be present and indicate the SRS processing and networking capabilities and requirements.

7.5 Metrics and Consumption Reporting

The Media Application Provider may use the Provisioning procedure to configure the collection of split rendering metrics and logging of consumption statistics. When present, this information shall be passed to the MSH using the Service Access Information procedure.

The SRC shall collect and report the data for a split rendering session that matches the criteria for metrics and/or consumption reporting as indicated by the configuration procedure.

8 Split Rendering User Plane

8.1 General

The user plane for split rendering covers all traffic between the SRC and SRS, or the SRC and any other Media AS. The common formats for split rendering are defined in this clause. Split rendering profiles may define additional user plane formats.

This clause illustrates the protocol stack for the User plane transport related to the signalling as well as the media delivery between SRC and SRS though M4.

<table>
<tr><td colspan="2">Signaling Path</td><td colspan="2">Media Path</td></tr>
<tr><td colspan="2">Session Negotiation and Setup</td><td>Data</td><td>Metadata</td></tr>
<tr><td>Configuration Format</td><td>Description Format</td><td>Media Formats</td><td>Metadata Formats</td></tr>
<tr><td colspan="2">SR Signaling Protocol</td><td>Media Data Stream Format</td><td>Metadata Channel Message Format</td></tr>
<tr><td colspan="2">SWAP (TS26.113)</td><td>Media Transport (RFC8834)</td><td>Data Channel (RFC8831)</td></tr>
<tr><td colspan="2">RTC-4s</td><td colspan="2">RTC-4m</td></tr>
</table>

Figure 8.1-1 Split rendering protocol Stack

8.2 Split Rendering Signalling Protocols

Both SRC and SRS shall support the SWAP protocol as defined in TS 26.113 [6] clause 13.2.

The SWAP protocol allows for the definition of application-specific messages.

The following application-specific messages shall be supported for split rendering:

- The configuration message carries the split rendering configuration information from the SRC to the SRS. It shall be identified by the type "**urn:3gpp:sr-mse:sr-configuration**" and the object shall be formatted according to clause 8.4.2.2.

- The rendering description message carries the description of the split rendered media from the SRS to SRC. The format of the message is SR-profile-specific and shall be defined by each profile. It shall be identified by the type "**urn:3gpp:sr-mse:sr-description**". The rendering description message provides the semantics of the media that is delivered over WebRTC from the SRS to SRC.

The SWAP message exchange for the establishment of a split rendering session is depicted by the following call flow diagram:

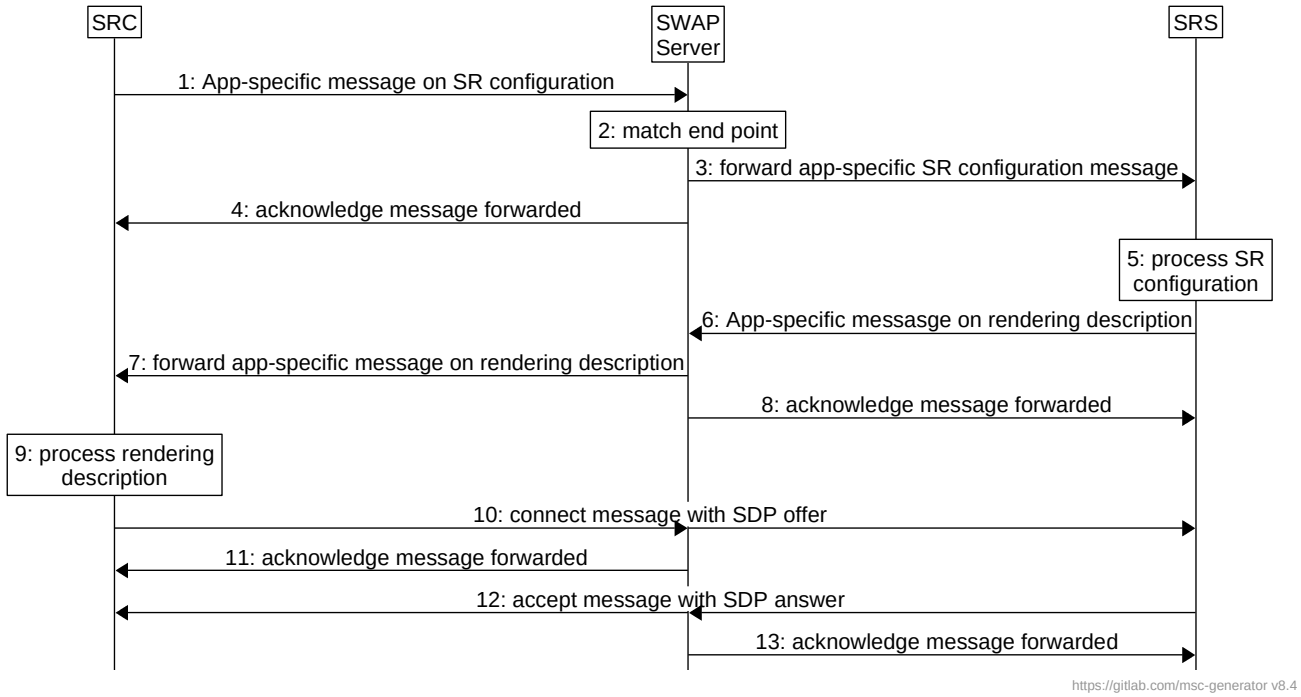

Figure 8.2-1 Call flows for SWAP message exchange

Pre-requisites:

- The SRC has discovered the identifier of the SRS that it will use for its split rendering session.

- The SRC has retrieved the address of the SWAP server as part of the configuration.

The stpes are as follows:

1. The SRC sends the configuration message as an application-specific SWAP message to the SWAP server. It provides the identifier of the target SRS as a matching criteria.

2. The SWAP server uses the provided matching criteria to locate the SRS.

3. The SWAP server forwards the configuration message to the target SRS.

4. The SWAP server confirms the successful forwarding of the message to the SRC.

5. The SRS processes the SR configuration message. It may for instance verify application and resource availablity, launch the application, configure its rendering, and create a rendering description.

6. The SRS sends the rendering description message as an application-specific SWAP message to the SWAP server.

7. The SWAP server forwards the message to the SRC.

8. The SWAP server acknowledges the successful forwarding of the message to the SRS.

9. The SRC processes the rendering description and identifies the required data channel and media sessions.

10. SRC sends a connect message with the SDP offer to the SRS. The offer reflects the negotiated media and data channel streams.

11. The SWAP server acknowledges the forwarding of the message to the SRS.

12. The SRS replies with an accept message that includes the SDP answer. The SDP answer reflects the information that was provided in the split rendering description.

13. The SWAP server acknowledges the forwarding of the message to the SRC.

8.3 Split Rendering Formats for Media and Metadata

8.3.1 General

This clause defines media and metadata formats that are common to one or more split rendering profiles.

8.3.2 Metadata Formats

8.3.2.1 General

Both SRC and SRS shall support the usage of the WebRTC data channel for the exchange of split rendering metadata. The WebRTC data channel shall declare “3gpp-sr” as the data channel sub-protocol. The message content format depends on the type of the message. The data channel sub-protocol is defined in clause 8.3.3.

Message types shall be unique identifiers in the URN format. This clause defines a set of message types and their formats. The messages are derived from the OpenXR API to ensure smooth operation with AR devices that support OpenXR. In case other XR APIs are used, mapping the message payload to the appropriate XR API structures shall be performed by the split rendering client.

8.3.2.2 Pose Format

The pose format that is used by all split rendering profiles defined by this specification shall comply with the format defined in TS 26.119 [4] clause 12.2. The pose information shall be carried as part of the data channel messaging mechanism defined in clause 8.3.3 and shall be provided in JSON format. The message type shall be “urn:3gpp:split-rendering:v1:pose”.

8.3.2.3 Action Format

The action information format that is used by all split rendering profiles defined by this specification shall comply with the format defined in TS 26.119 [4] clause 12.3. The action information shall be carried as part of the data channel messaging mechanism defined in clause 8.3.3 and shall be provided in JSON format. The message type shall be “urn:3gpp:split-rendering:v1:action”.

8.3.3 Metadata Data Channel Message Format

For the carriage of metadata defined in clause 8.3, such as pose and action information, the SRS and SRC shall use the WebRTC data channel. The data channel sub-protocol shall be identified as “3gpp-sr-metadata”, which shall be included in the dcmap attribute of the SDP.

The transmission order for the data channel shall be set to in-order and the transmission reliability shall be set to reliable.

The split rendering metadata message format shall be set to text-based and the messages shall be UTF-8 encoded JSON messages.

A data channel message may carry one or more split rendering messages as defined in Table 8.3.3-1.

Table 8.3.3-*1* Split Rendering Metadata Messages Format

Name	Type	Cardinality	Description
messages	Array(Message)	1..n	A list of split rendering metadata messages. Each message shall be formatted according to the Message data type as defined in Table 8.3.3-2.

Each split rendering message shall follow the format specified in Table 8.3.3-2.

Table 8.3.3-2 Split Rendering Metadata Message Data Type

Name	Type	Cardinality	Description
id	string	1..1	An unique identifier of the message in the scope of the data channel session.
Type	string	1..1	A urn that identifies the message type.
Message	object	1..1	The message content depends on the message type.
sendingAtTime (ref. T1')	number	0..1	The time when the split rendering metadata message is transmitted from the split rendering client to the split rendering server.

8.4 Split Rendering Formats for Session Setup and Negotiation

8.4.1 General

In Figure 5.2.1.2-1, in step 5, the negotiation between the SRC and SRS for the split-rendering configuration takes place. The detailed call flow for such a negotiation between the SRC and the SRS may vary. Depending upon the split rendering profile, the negotiation between the SRC and the SRS may be straight forward or go back and forth.

In the simplest case, the SRC provides SRS the capabilities of the device and if SRS can accommodate the split-rendering processing that addresses the device's needs and capabilities, it confirms by providing a description of the output format. In such scheme, the SRS is responsible to make the decision and no back-and-forth negotiation occurs.

8.4.2 Split Rendering Configuration Format

8.4.2.1 Introduction

The Split Rendering client establishes an XR session locally based on the device configuration and user selection. The SR client defines the view configuration (e.g. mono or stereo views), the projection format (such as projection, equirectangular, quad, or cubemap), the swap chain image configuration, etc.

In addition, XR space and action configurations are negotiated between the SR client and server. This includes defining common XR spaces and defining and selecting actions and action sets.

The format is extensible to support the exchange of additional/future configuration information.

8.4.2.2 Split Rendering Configuration Format

The session configuration information shall be in JSON format. It shall have the following format:

Table 8.4.2.2-1 Split Rendering Configuration Format

Name	Type	Cardinality	Description
renderingFlags	Array(SR_CONFIG_FLAGS)	0..1	Provides a set of flags to activate/deactivate selected rendering functions. The defined SR_CONFIG_FLAGS are: - FLAG_ALPHA_BLENDING - FLAG_DEPTH_COMPOSITION - FLAG_EYE_GAZE_TRACKING
splitRenderingProfile	array(URI)	0..1	A list of supported split-rendering profile identifiers on the UE. The profile identifiers are listed in Annex C for each profile.
deviceCapabilities	Object	0..1	Device capabilities as defined in TS 26.119 [4], clause 6.2.5.
spaceConfiguration	Object	0..1	The space configuration is typically sent by the split rendering server to the split rendering client. Upon reception of this information, the SR client uses this information to create the reference and action spaces as well as to agree on common identifiers for the XR spaces.
referenceSpaces	Array	0..1	An array of reference spaces and their identifiers.
id	number	1..1	A unique identifier of the XR space in the context of the split rendering session.
refSpace	enum	1..1	One of the defined reference spaces in OpenXR. These may be: XR_REFERENCE_SPACE_TYPE_VIEW, XR_REFERENCE_SPACE_TYPE_LOCAL, or XR_REFERENCE_SPACE_TYPE_STAGE.
actionSpaces	Array	0..1	An array of action spaces that need to be defined by the split rendering client in the XR session.
id	number	1..1	A unique identifier of the XR space in the context of the split rendering session.
actionId	number	1..1	Provides the unique identifier of the action.
subactionPath	string	1..1	The subaction path identifies the action, which can then be mapped by the XR runtime to user input modalities.
initialPose	Pose	0..1	Provides the initial pose of the new XR space’s origin.

viewConfiguration	Object	0..1	Conveys the view configuration that is configured for the XR session.
type	Enum	1..1	The type indicates the view configuration. Defined values are MONO and STEREO. Other values may be added.
width	number	1..1	The recommended width of the swapchain image.
height	number	1..1	The recommended height of the swapchain image.
compositionLayer	string	1..1	An identifier of the selected composition layer.
minPoseInterval	number	0..1	The minimum time interval between two consecutive pose information instances sent to the network, in milliseconds.
fovs	Array	0..1	An array that provides a list of the field of views (FoV) associated with each view.
fov	Object	1..n	Indicates the four sides of the field of view used for the projection of the corresponding XR view. The number of views n is determined by the *type* enum of the *viewConfiguration*. Both the *viewPoses* in the Pose Format and the *fovs* arrays shall be ordered in a consistent way (i.e., a same index can be used to retrieve the view pose and the related FoV information).
angleLeft	number	1..1	The angle of the left side of the field of view. For a symmetric field of view this value is negative.
angleRight	number	1..1	The angle of the right side of the field of view.
angleUp	number	1..1	The angle of the top part of the field of view.
angleDown	number	1..1	The angle of the bottom part of the field of view. For a symmetric field of view this value is negative.
environmentBlendMode	enum	1..1	The type indicates the environment blend mode configuration. Defined values are OPAQUE, ADDITIVE and ALPHA_BLEND. Other values may be added.
actionConfiguration	Array	0..1	This contains a list of the actions that are to be defined by the SR client.
action	Object	1..n	A definition of a single action object.
id	number	1..1	A unique identifier of the action.
actionType	enum	1..1	The type of the action state. This can be a Boolean, float, vector2, pose, vibration output, etc.

subactionPaths	string	1..n	An array of subaction paths associated with this action. The split rendering client will provide the state of all defined sub-action paths.
extraConfigurations	Object	0..1	A placeholder for addition configuration information.

8.4.3 Output Format Description

The output format description depends on the split rendering profile and is defined by the split rendering profile in Annex C.

8.5 Split Rendering Transport Protocols

Split Rendering shall use WebRTC for the real-time transport of the rendered media. The RTP restrictions for WebRTC as specified in RFC8834 shall apply. The usage of the WebRTC data channel shall be in accordance with RFC8831.

Editor's Note: applicable guidelines for the usage of the PDU Set Marking are pending completion of TS 26.522 [8].

9 Split Rendering Client

9.1 Functionality

The Split Rendering Client (SRC) is a function that runs on the UE to provide split rendering functionality to applications. The SRC is designed to be offered as an SDK to application developers. The SRC abstracts the details of the split rendering operation and provides a simple to use API to the application to facilitate the usage of split rendering.

The SRC performs the following functions:

- Creates and manages the XR session,
- Discovers and selects a split rendering server (SRS) in the network,
- Establishes a split rendering session with the SRS,
- Communicates the necessary information about the session to the MSH/AF to benefit from dynamic policy, network assistance, consumption reporting, etc.
- Operates the rendering loop on the UE.

9.2 Client API

As described in clause 5.1.3, the SRC exposes an API over M7 interface to the application. The SRC defines the following interface:

Table 9.2-1 Split Rendering Client API

Method	Parameters		State after Success	Description
	in	out		
SplitRenderer()	- appId - aspId? - externalServiceId? - preferences?	- srSessionId	STATE_PROCESSING	Creates a SplitRenderer instance, representing the SRC, which can subsequently be used to connect to an SRS and perform split rendering.
getState()	- srSessionId	- state	N/A	Returns the current state of the SRC. Possible states are: STATE_IDLE, STATE_PROCESSING, STATE_READY, STATE_RUNNING, STATE_STOPPED.
getConfiguration()	- srSessionId	- configuration	N/A	Allows the application to query the current configuration of the split rendering session.
start()	- srSessionId	- boolean	STATE_RUNNING	Instructs the SRC to discover and connect to an SRS. If current state is not STATE_READY, the connection will fail.
stop()	- srSessionId - reason?		STATE_STOPPED	Terminates the connection to the SRS.
release()	- srSessionId		STATE_IDLE	Releases all resources associated with the Split Rendering session.
getMetrics()	- srSessionId - metrics	- metrics report	N/A	Retrieves a set of metric reports for the split rendering session that describe the quality of experience of the session.

The application is able to subscribe to events related to the split rendering session by setting the corresponding event handler.

The supported events are:

- State change: the state of the SR session has changed
- Error: an error has occurred during the split rendering session. The error is not severe enough to cause a state change to the STATE_ERROR state.
- Quality change: the SRC has observed a change in the quality of the split rendering session. This may involve one or more SR metrics.

The Preferences object shall contain the following information:

- Information about the desired rendering, e.g. choose to render on 2D device or on one of the available connected XR devices.

The criteria object may contain the following information:

- Requirements for latency and bitrate that are different from the ones in the provisioning,
- KPIs for the SRS instance, such as its graphics capabilities or current load.

The parameters are defined as follows:

- aspId: a string that holds an identifier of the application service provider. The value is provisioned by the application service provider as defined in TS 26.510[9].
- appId: a string that holds an identifier of the application. This value is provisioned by the application service provider as defined in TS 26.510[9].
- externalServiceId: An identifier assigned by the Service Application Provider and shared with the SRC over M8 as defined in TS 26.510[9].
- preferences: the preferences object carries parameters about the user's current preferences. These include the preferred display configuration, e.g. 2D display, HMD, etc. It may also include information about quality versus latency preferences.
- configuration: the configuration object stores information about the currently active configuration for the session. It carries the same information as described in clause 8.4.2.2.
- srSessionId: the srSessionId is a unique identifier of the split rendering session at the SRC.
- metrics: the metrics and metrics report objects provide the current status of a selected set of metrics that pertain to the current split rendering session. The format should follw clause 7.5.

9.3 Split Rendering Metrics

9.3.1 General

This clause defines a set of metrics that are relevant to the operation of a split rendering session.

9.3.2 QoE Metrics Formats

9.3.2.1 Timing Information Format

The timing information associated with the rendered frame is transmitted in the RTCP report block formats. This timing information is listed in the Table 9.3.22-1.

The SRS may use the "QoE timing information" RTCP Extended Reports messages to transmit the timing information required for measuring the QoE metrics to an SRC. The RTCP report block format for transmitting the QoE timing information is specified **in TS 26.522 [8]**. SDP signalling required for negotiating the transmission of QoE metrics between the UE and the SRS is documented **in TS 26.522 [8]**.

The latency metrics that use the timing information defined in Table 9.3.2-1 are detailed in the section 9.3.2.2.

Table 9.3.2.1-1: Timing information in the RTCP block formats.

Name	Description
estimatedAtTime (ref. T1)	This wall clock time is defined in TS 26.119 [4] - Pose Format. This time is sent from the split rendering client. This time is then received by the split rendering server and sent back to the split rendering client with the associated media frame.

sendingAtTime (ref. T1')	This time is defined in Table 8.32.3-2 - Split Rendering Metadata Message Data Type This time is sent from the split rendering client. This time is then received by the split rendering server and sent back to the split rendering client with the associated media frame.
startToRenderAtTime (ref. T3)	The time when the renderer in the Split Rendering Server starts to render the associated media frame.
sceneUpdateTime (ref. T6)	The time when the Scene Manager starts to update the 3D scene graph according to the viewer pose and the user actions.
serverTransmitTime (ref. T5)	The time when the encoded rendered frame is transmitted from the split rendering server to the split rendering client.

9.3.2.2 Latency metrics

To enable good XR experiences, it is relevant to monitor latencies such as the motion-to-photon and the pose-to-render-to-photon.

Beyond the sense of presence and immersiveness, the age of the content and user interaction delay are of the uttermost importance for immersive and non-immersive interactive experiences, i.e., experiences for which the user interaction with the scene impacts the content of scene (such as online gaming).

Table 9.3.2.2-1 provides timing information collected to compute the latency metrics at the split rendering client or split rendering server endpoint.

Table 9.3.2.2-1: Timing information for latency metrics

Source endpoint	Timing information	Definition
Split Rendering Client	estimatedAtTime (ref. T1)	This time is expressed in wall clock time (refer to Table 9.3.2-1).
Split Rendering Client	lastChangeTime	The time the user action is made. It corresponds to the lastChangeTime field defined in the action format in Table 9.3.2-1. This time is expressed in XR system time clock.
Split Rendering Server	sceneUpdateTime (ref. T6)	This time is a NTP timestamp format (refer to Table 9.3.2-1).
Split Rendering Server	startToRenderAtTime (ref. T3)	This time is expressed in wall clock time (refer to Table 9.3.2-1).
Split Rendering Client	actualDisplayTime (ref. T2.actual)	The actual display time of the rendered frame in the swapchain. The estimation of the actual display time is available through the XR runtime.
Split Rendering Client	sendingAtTime (ref. T1')	This time is expressed in wall clock time (refer to Table 9.3.2-1).
Split Rendering Server	serverTransmitTime (ref. T5)	This time is expressed in wall clock time (refer to Table 9.3.2-1).

Split Rendering Client	receptionTime	The time when the data is received by the split rendering client.

The latency metrics are specified in Table 9.3.2.2-2. Latency calculation formulas are defined using the timing information defined in Table 9.3.2.2-1.

Table 9.3.2.2-2: Latency metrics

Latency metric	**Description**
poseToRenderToPhoton	The time duration, in units of milliseconds, between the time to provide the pose information from the XR runtime to the renderer (the renderer uses this pose to generate the rendered frame) and the display time of the rendered frame. It can be computed as follows: actualDisplayTime – estimatedAtTime
renderToPhoton	The time duration, in units of milliseconds, between the start of the rendering by the Presentation Engine and the display time of the rendered frame. It can be computed as follows: actualDisplayTime – startToRenderAtTime (NOTE 1)
roundtripInteractionDelay	The time duration, in units of milliseconds, between the time a user action is initiated and the time the action is presented to the user. It can be computed as follows: actualDisplayTime – lastChangeTime
userInteractionDelay	The time duration, in units of milliseconds, between the time a user action is initiated and the time the action is taken into account by the content creation engine in the scene manager. It can be computed as follows: sceneUpdateTime – lastChangeTime (NOTE 1)
ageOfContent	The time duration, in units of milliseconds, between the time the content is created in the scene by the Scene Manager and the time it is presented to the user. It can be computed as follows: actualDisplayTime – sceneUpdateTime (NOTE 1)
sceneUpdateDelay	The time duration, in units of milliseconds, spent by the Scene Manager to update the scene graph. It can be computed as follows: startToRenderAtTime – sceneUpdateTime
metadataDelay	The time duration, in units of milliseconds, between the time the split rendering metadata message is sent from the split rendering client and the time the split rendering server start to render using that metadata.

	It can be computed as follows: startToRenderAtTime – sendingAtTime
dataFrameDelay	The time duration, in units of milliseconds, spent to transmit the media rendered frame from the split rendering server to the split rendering client. It can be computed as follows: receptionTime – serverTransmitTime
NOTE 1: for the latency metrics computation, the timing information mentioned above need to be converted to a single time format (e.g., Wall clock time).	

9.3.3 QoE Metrics reporting protocol

The Metrics Reporting API allows the Media Session Handler to send QoE metrics reports to the metrics collection server.

An SR UE supporting Quality of Experience shall report QoE metrics according to the QoE configuration. QoE reporting is optional, but if an MSH reports QoE metrics, it shall report all requested metrics.

9.3.4 QoE metrics definition

9.3.4.1 Introduction

This clause extends clause 15.2 of TS 26.113 that provides the general QoE metric definitions and measurement framework. An SR UE supporting the QoE metrics feature shall support the reporting of the metrics in this clause.

The metrics are calculated for each measurement resolution interval "`measureinterval`" as described in clause 15.2.1 of TS 26.113. They are reported to the server according to the reporting interval "`reportinginterval`" and after the end of the session as described in clause 15.2.1 of TS 26.113.

9.3.4.2 Pose to render to photon metric

The *PoseToRenderToPhoton* duration is the time duration between the time at which the pose information is available from the XR runtime to the renderer and the display time of the rendered frame. The unit of this metric is expressed in milli seconds.

The average pose to render to photon is equal to the sum of *PoseToRenderToPhoton* duration of each frame during the measurement resolution period divided by the time duration, in seconds, of the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value.

The minimum pose to render to photon duration is equal to the lowest value of *PoseToRenderToPhoton* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum pose to render to photon duration is equal to the highest value of *PoseToRenderToPhoton* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The syntax for the metric "*PoseToRenderToPhoton*" metric is as defined in Table 9.3.4.2-1.

Table 9.3.4.2-1: Pose to render to photon metric information for Quality Reporting

Key		Type	Description
PoseToRenderToPhoton		Object	
	@avgPoseToRenderToPhoton	doubleVectorType	An unordered list of all average pose to render to photon delay measured within each measurement resolution period.
	@minPoseToRenderToPhoton	unsignedIntVectorType	The minimum pose to render to photon duration is equal to the lowest value of PoseToRenderToPhoton duration measured during each measurement resolution period. Provides an unordered list of minimum pose to render to photon delay measured during a metric reporting period.
	@maxPoseToRenderToPhoton	unsignedIntVectorType	The maximum pose to render to photon duration is equal to the highest value of PoseToRenderToPhoton duration measured during each measurement resolution period. Provides an unordered list of maximum pose to render to photon delay measured during a metric reporting period.

9.3.4.3 Render to photon metric

The *renderToPhoton* duration is the time duration between the time at which the presentation engine started rendering to the display time of the rendered frame. The unit of this metric is expressed in milli seconds.

The average render to photon metric is equal to the sum of *renderToPhoton* duration of each frame during the measurement resolution period divided by the time duration, in seconds, of the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value.

The minimum render to photon duration is equal to the lowest value of *renderToPhoton* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum render to photon duration is equal to the highest value of *renderToPhoton* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The syntax for the metric "*renderToPhoton*" metric is as defined in Table 9.3.4.3-1.

Table 9.3.4.3-1: Render to Photon metric information for Quality Reporting

Key		Type	Description
renderToPhoton		Object	
	avgRenderToPhoton	doubleVectorType	An unordered list of all average render to photon delay measured within each measurement resolution period.
	@minRenderToPhoton	unsignedIntVectorType	The minimum render to photon duration is equal to the lowest value of renderToPhoton duration measured during each measurement resolution period. Provides an unordered list of minimum render to photon delay measured during a metric reporting period.
	@maxRenderToPhoton	unsignedIntVectorType	The maximum render to photon duration is equal to the highest value of renderToPhoton duration measured during each measurement resolution period. Provides an unordered list of maximum render to photon delay measured during a metric reporting period.

9.3.4.4 Round-trip interaction delay metric

The *roundtripInteractionDelay* duration is the time duration between the time a user action is initiated to the time the action is presented to the user. The unit of this metric is expressed in milli seconds.

The average round trip interaction delay metric is equal to the sum of *roundtripInteractionDelay* duration of each action during the measurement resolution period divided by the number of actions in the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value. Within each resolution period the number of user actions are summed up and stored in the vector *@numberOfInteractionEvents.*

The minimum round trip interaction delay duration is equal to the lowest value of *roundtripInteractionDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum round trip interaction delay duration is equal to the highest value of *roundtripInteractionDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The identifier of a user action with minimum, and maximum, round trip interaction delays within each measurement resolution period are provided in the *@minActionIDs*, *@maxActionIDs* respectively, as an unordered list of user action identifiers.

The syntax for the metric "*roundtripInteractionDelay*" metric is as defined in Table 9.3.4.4-1

Table 9.3.4.4-1: Round-trip interaction delay metric information for Quality Reporting

Key		Type	Description
`roundtripInteractionDelay`		`Object`	
	`@avgRoundTripInteractionDelay`	`doubleVectorType`	An unordered list of all average round trip interaction delays measured within each measurement resolution period.
	`@numberOfUserActions`	`unsignedIntVectorType`	The number of user actions within each measurement resolution period are summed up and stored in the vector. Provides an unordered list of user actions (occurred within each measurement period) measured during a metric reporting period.
	`@minRoundTripInteractionDelay`	`unsignedIntVectorType`	The minimum round trip interaction delay duration is equal to the lowest value of `roundtripInteractionDelay` duration measured during each measurement resolution period. Provides an unordered list of minimum round trip interaction delay measured during a metric reporting period.
	`@minActionIDs`	`unsignedIntVectorType`	The identifier of a user action with minimum round trip interaction delay within each measurement resolution period. Provides an unordered list of user action identifiers (occurred within each measurement period) with minimum round trip interaction delay measured during a metric reporting period.
	`@maxRoundTripInteractionDelay`	`unsignedIntVectorType`	The maximum round trip interaction delay duration is equal to the highest value of `roundtripInteractionDelay` duration measured during each measurement resolution period. Provides an unordered list of maximum round trip interaction delay measured during a metric reporting period.
	`@maxActionIDs`	`unsignedIntVectorType`	The identifier of a user action with maximum round trip interaction delay within each measurement resolution period. Provides an unordered list of user action identifiers (occurred within each measurement period) with maximum round trip interaction delay measured during a metric reporting period.

9.3.4.5 User interaction delay metric

The *userInteractionDelay* duration is the time duration between the time a user action is initiated to the time the action is taken account by the content creation engine in the scene manager. The unit of this metric is expressed in milli seconds.

The average user interaction delay metric is equal to the sum of *userInteractionDelay* duration of each action during the measurement resolution period divided by the number of actions in the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value.

The minimum user interaction delay duration is equal to the lowest value of *userInteractionDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum user interaction delay duration is equal to the highest value of *userInteractionDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The identifier of a user action with minimum and maximum user interaction delays within each measurement resolution period are provided in the *@minActionIDs*, and *@maxActionIDs* respectively as an unordered list of user action identifiers.

The syntax for the metric "*userInteractionDelay*" metric is as defined in Table 9.3.4.5-1.

Table 9.3.45-1: User interaction delay metric information for Quality Reporting

Key		Type	Description
`userInteractionDelay`		`Object`	
	`@avgUserInteractionDelay`	`doubleVectorType`	An unordered list of all average user interaction delays measured within each measurement resolution period.
	`@numberOfUserActions`	`unsignedIntVectorType`	The number of user actions within each measurement resolution period are summed up and stored in the vector. Provides an unordered list of user actions (occurred within each measurement period) measured during a metric reporting period.
	`@minUserInteractionDelay`	`unsignedIntVectorType`	The user interaction delay duration is equal to the lowest value of `userInteractionDelay` duration measured during each measurement resolution period. Provides an unordered list of user interaction delay measured during a metric reporting period.
	`@minActionIDs`	`unsignedIntVectorType`	The identifier of a user action with minimum round trip interaction delay within each measurement resolution period. Provides an unordered list of user action identifiers (occurred within each measurement period) with minimum round trip interaction delay measured during a metric reporting period.
	`@maxUserInteractionDelay`	`unsignedIntVectorType`	The maximum user interaction delay duration is equal to the highest value of `userInteractionDelay` duration measured during each measurement resolution period. Provides an unordered list of maximum user interaction delay measured during a metric reporting period.
	`@maxActionIDs`	`unsignedIntVectorType`	The identifier of a user action with maximum round trip interaction delay within each measurement resolution period. Provides an unordered list of user action identifiers (occurred within each measurement period) with maximum round trip interaction delay measured during a metric reporting period.

9.3.4.6 Age of contents metric

The *ageOfContent* is the time duration between the time the content is created in the scene by the scene manager and the time it is presented to the user. The unit of this metric is expressed in milli seconds. Within each measurement resolution period the number of scene creations and updates are counted and stored in the vector *@numberOfSceneEvents.*

The average age of content metric is equal to the sum of *ageOfContent* durations of all scene creation and update events during the measurement resolution period divided by the total number of scene creation and updates in the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value.

The minimum age of content duration is equal to the lowest value of *ageOfContent* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum age of content duration is equal to the highest value of *ageOfContent* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The syntax for the metric "*ageOfContent*" metric is as defined in Table 9.3.2.6-1.

Table 9.3.2.6-1: Age of Content metric information for Quality Reporting

Key		Type	Description
`ageOfContent`		`Object`	
	`@avgAgeOfContent`	`doubleVectorType`	An unordered list of average age of content durations measured within each measurement resolution period.
	`@numberOfSceneEvents`	`unsignedIntVectorType`	The number of scene creation and/or scene updates within each measurement resolution period are stored in the vector. Provides an unordered list of scene creation and/or scene updates (occurred within each measurement period) measured during a metric reporting period.
	`@minAgeOfContent`	`unsignedIntVectorType`	The minimum age of content duration is equal to the lowest value of `ageOfContent` measured during each measurement resolution period. Provides an unordered list of minimum age of content measured during a metric reporting period.
	`@maxAgeOfContent`	`unsignedIntVectorType`	The maximum age of content is equal to the highest value of `ageOfContent` measured during each measurement resolution period. Provides an unordered list of maximum age of content measured during a metric reporting period.

9.3.2.7 Scene update delay metric

The *sceneUpdateDelay* duration is the time duration spent by the scene manager to update the scene graph. The unit of this metric is expressed in milliseconds. Within each measurement resolution period the number of scene updates are counted and stored in the vector *@numberOfSceneUpdates.*

The average scene update delay duration metric is equal to the sum of *sceneUpdateDelay* durations of all scene updates during the measurement resolution period divided by the total number of scene updates in the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value.

The minimum scene update delay is equal to the lowest value of *sceneUpdateDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum age of content duration is equal to the highest value of *sceneUpdateDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The syntax for the metric "*sceneUpdateDelay*" metric is as defined in Table 9.3.2.7-1.

Table 9.3.2.7-1: Scene update delay metric information for Quality Reporting

Key		Type	Description
sceneUpdateDelay		Object	
	@avgSceneUpdateDelay	doubleVectorType	An unordered list of average scene update delays measured within each measurement resolution period.
	@numberOfSceneupdates	unsignedIntVectorType	The number of scene updates within each measurement resolution period are stored in the vector. Provides an unordered list of scene updates (occurred within each measurement period) measured during a metric reporting period.
	@minSceneUpdateDelay	unsignedIntVectorType	The minimum scene update delay is equal to the lowest value of sceneUpdateDelay measured during each measurement resolution period. Provides an unordered list of minimum scene update delay duration measured during a metric reporting period.
	@maxSceneUpdateDelay	unsignedIntVectorType	The maximum age of content is equal to the highest value of sceneUpdateDelay measured during each measurement resolution period. Provides an unordered list of maximum scene update delay duration measured during a metric reporting period.

9.3.2.8 Metadata delay metric

The *metadataDelay* duration is the time duration between the time the split rendering metadata is sent from the SRC and the time the split rendering server start to render using that metadata. The unit of this metric is expressed in milliseconds.

The average metadata delay metric is equal to the sum of *metadataDelay* duration of each metadata message during the measurement resolution period divided by the number of metadata messages in the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value. Within each measurement resolution period the number of metadata messages are summed up and stored in the vector *@numberOfMetadataMessages.*

The minimum metadata delay is equal to the lowest value of *metadataDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum metadata delay is equal to the highest value of *metadataDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The syntax for the metric "*metadataDelay*" metric is as defined in Table 9.3.2.8-1.

Table 9.3.2.8-1: Metadata delay metric information for Quality Reporting

Key		Type	Description
metadataDelay		Object	
	@avgmetadataDelay	doubleVectorType	An unordered list of average metadata delays measured within each measurement resolution period.
	@numberOfMetadata Messages	unsignedIntVectorType	The number of metadata messages within each measurement resolution period are stored in the vector. Provides an unordered list of total number of metadata messages (occurred within each measurement period) measured during a metric reporting period.
	@minMetadaDataDel ay	unsignedIntVectorType	The minimum metadata delay is equal to the lowest value of metadataDelay measured during each measurement resolution period. Provides an unordered list of minimum metadata delay duration measured during a metric reporting period.
	@maxMetadataDelay	unsignedIntVectorType	The maximum metadata message delay is equal to the highest value of metadataDelay measured during each measurement resolution period. Provides an unordered list of maximum metadata delay duration measured during a metric reporting period.

9.3.2.9 Data frames delay metric

The *dataFrameDelay* duration is the time duration between the time the media rendered frame is transmitted from the split rendering server to the time the split rendering client received the data frame. The unit of this metric is expressed in milliseconds.

The average data frame delay metric is equal to the sum of *dataFrameDelay* duration of each transmitted frame during the measurement resolution period divided by the number of frames transmitted in the measurement resolution period. The unit of this metric is expressed in milli seconds and can be a fractional value. Within each measurement resolution period the number of data frames transmitted are summed up and stored in the vector *@numberOfDataFrames.*

The minimum data frame delay is equal to the lowest value of *dataFrameDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The maximum data frame delay is equal to the highest value of *dataFrameDelay* duration measured during the measurement resolution period. The unit of this metric is expressed in milli seconds and is an integer value.

The syntax for the metric "*dataFrameDelay*" metric is as defined in Table 9.3.2.9-1.

Table 9.3.2.9-1: Data frame delay metric information for Quality Reporting

Key		Type	Description
dataFrameDelay		Object	
	@avgdataFrameDelay	doubleVectorType	An unordered list of average data frame transmission delays measured within each measurement resolution period.
	@numberOfDataFrames	unsignedIntVectorType	The number of data frames transmitted within each measurement resolution period are stored in the vector. Provides an unordered list of total number of data frames transmitted (within each measurement period) during a metric reporting period.
	@minDataFrameDelay	unsignedIntVectorType	The minimum data frame delay is equal to the lowest value of dataFrameDelay measured during each measurement resolution period. Provides an unordered list of minimum data frame delay duration measured during a metric reporting period.
	@maxdataFrameDelay	unsignedIntVectorType	The maximum data frame delay is equal to the highest value of dataFrameDelay measured during each measurement resolution period. Provides an unordered list of maximum data frame delay duration measured during a metric reporting period.

9.3.5 Quality metrics reporting

9.3.5.1 General

The quality metrics report follows the XML-based report format defined in clause 9.3.5.2.

The MIME type of an XML-formatted QoE report shall be "`application/3gprtc-qoe-report+xml`".

The metrics reporting protocol is as defined in clause 9.5.3 of TS 26.510. Split rendering UEs shall use the above MIME content type. The metrics report format is defined in the following sub clause.

9.3.5.2 Report format

The QoE report is formatted as an XML document that complies with the XML schema in listing 10.6.2-1 of TS 26.247 [17].

The schema in listing 9.3.5.2-1 is an extension to allow additional QoE metrics for SR UE to be reported using the QoE report specified in clause 10.6.2 of TS 26.247 [17]. The filename of this schema is "TS26565_SR_MSEQoEMetrics.xsd".

Listing 9.3.5.2-1: SR_MSE QoE Metrics XML schema

```
<?xml version="1.0"?>
<xs:schema version="TSG104-Rel18" xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:3gpp:metadata:2024:RTC:SR_MSEQoEMetrics"
    xmlns:sv="urn:3gpp:metadata:2016:PSS:schemaVersion"
    xmlns="urn:3gpp:metadata:2024:RTC:SR_MSEQoEMetrics" elementFormDefault="qualified">


    <xs:element name="QoeMetric" type="QoeMetricType"/>

    <xs:complexType name="QoeMetricType">
        <xs:sequence>
            <xs:choice>
                <xs:element name="poseToRenderToPhoton" type="PoseToRenderToPhotonType"/>
                <xs:element name="renderToPhoton" type="RenderToPhotonType"/>
                <xs:element name="roundTripInteractionDelay"
type="RoundTripInteractionDelayType"/>
                <xs:element name="userInteractionDelay" type="UserInteractionDelayType"/>
                <xs:element name="ageOfContent" type="AgeOfContentType"/>
```

```
                <xs:element name="sceneUpdateDelay" type="SceneUpdateDelayType"/>
                <xs:element name="metadataDelay" type="MetadataDelayType"/>
                <xs:element name="dataFrameDelay" type="DataFrameDelayType"/>
            </xs:choice>
            <xs:element ref="sv:delimiter"/>
            <xs:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="PoseToRenderToPhotonType">
        <xs:attribute name="avgPoseToRenderToPhoton" type="doubleVectorType" use="required"/>
        <xs:attribute name="minPoseToRenderToPhoton" type="unsignedIntVectorType"
use="required"/>
        <xs:attribute name="maxPoseToRenderToPhoton" type="unsignedIntVectorType"
use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="RenderToPhotonType">
        <xs:attribute name="avgPoseToRenderToPhoton" type="doubleVectorType" use="required"/>
        <xs:attribute name="minRenderToPhoton" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxRenderToPhoton" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="RoundTripInteractionDelayType">
        <xs:attribute name="avgRoundTripInteractionDelay" type="doubleVectorType"
use="required"/>
        <xs:attribute name="numberOfUserActions" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="minRoundTripInteractionDelay" type="unsignedIntVectorType"
use="required"/>
        <xs:attribute name="minActionIds" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxRoundTripInteractionDelay" type="UnsignedIntVectorType"
use="required"/>
        <xs:attribute name="maxActionIds" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="UserInteractionDelayType">
        <xs:attribute name="avgUserInetractionDelay" type="doubleVectorType" use="required"/>
        <xs:attribute name="numberOfUserActions" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="minUserInetractionDelay" type="unsignedIntVectorType"
use="required"/>
        <xs:attribute name="minActionIds" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxUserInteractionDelay" type="unsignedIntVectorType"
use="required"/>
        <xs:attribute name="maxActionIds" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="AgeOfContentType">
        <xs:attribute name="avgAgeOfContent" type="doubleVectorType" use="required"/>
        <xs:attribute name="mumberOfSceneEvents" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="minageOfContent" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxAgeOfContent" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="SceneUpdateDelayType">
        <xs:attribute name="avgSceneUpdateDelay" type="doubleVectorType" use="required"/>
        <xs:attribute name="numberOfSceneUpdates" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="minsceneUpdateDelay" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxsceneUpdateDelay" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>

    <xs:complexType name="MetadataDelayType">
        <xs:attribute name="avgMetadataDelay" type="doubleVectorType" use="required"/>
        <xs:attribute name="numberOfMetadataMessages" type="unsignedIntVectorType"
use="required"/>
        <xs:attribute name="minMetadataDelay" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxMetadataDelay" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>
```

```
<xs:complexType name="DataFrameDelayType">
        <xs:attribute name="avgDataFrameDelay" type="doubleVectorType" use="required"/>
        <xs:attribute name="numberOfDataFrames" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="minDataFrameDelay" type="unsignedIntVectorType" use="required"/>
        <xs:attribute name="maxDataFrameDelay" type="unsignedIntVectorType" use="required"/>
        <xs:anyAttribute processContents="skip"/>
    </xs:complexType>



    <xs:simpleType name="doubleVectorType">
        <xs:list itemType="xs:double"/>
    </xs:simpleType>



    <xs:simpleType name="unsignedIntVectorType">
        <xs:list itemType="xs:unsignedInt"/>
    </xs:simpleType>

</xs:schema>
```

9.3.5.3 Quality Reporting Scheme and Metrics reporting configuration for SRC

An SR UE shall use the metrics reporting scheme defined in clause 6.7 of TS 26.113 [6]. The Metrics Reporting Provisioning API allows an RTC Application Provider to configure the Metrics Collection and Reporting procedure for a particular split rendering session at reference point RTC-1. The Service Access Information API allows an RTC Application Function to configure the metrics collection and reporting process for a particular split rendering session at reference point M5.

An SR UE shall use the data model for metrics reporting provisioning API defined in clause 8.10.3.1 of TS 26.510 [9] and the Service Access Information API defined in clause 9.2.3 of TS 26.510 [9]. The *metrics* element present in the *MetricsReportingConfiguration* resource and the *ServiceAccessInformation* resource shall include zero or more metrics defined in clause 9.3.4 of this document in addition to the quality metrics defined in clause 15.2 of TS 26.113 [6].

10 Security and Privacy Aspects

10.1 Security

Signaling for session establishment and exchange of application-specific messages shall use a secure transport channel based on WebSockets as defined in TS 26.113 [6].

Media transport shall be secured by the usage of WebRTC.

10.2 Privacy

Users of the split rendering MSE shall be aware that the application data and traffic are fully accessible to the SRS. The SRC shall ensure that the SRS used is trusted by the Media Application Provider, for example through the validation of the SRS's X.509 certificates.

Annex A (informative): Implementation Guidelines

A.1 Guidelines for Application Developers

Application developers may use the SR_MSE enabler as an SDK for developing applications that benefit from Split Rendering.

The SDK may be accessible through an API that conforms to the API definition in 9.2.

Application developers should implement monitoring of the split rendering session quality in their applications and always be aware that not all functionality described in this specification is always available for all split rendering sessions. It is then up to the application to decide whether the usage of split rendering is acceptable or not.

A.2 Guidelines for Split Rendering MSE Implementers

A.2.1 Guidelines for implementers of the Split Rendering Server

If the use of eye gaze tracking is activated, the SRS may use this confidence information to perform gaze-based optimizations like foveated rendering and foveated video encoding. In foveated rendering, the rendering engine renders areas of a picture with higher quality than others to match the user's current gaze, while in foveated video encoding areas of the picture are encoded with a higher SNR quality than other areas, to match the user's current gaze.With gaze predictions, the SRS should create an importance map for the picture based on the confidence values associated with the gaze predictions. Additionally, the SRS may also use other information to produce the importance map, such as content Regions of Interest, type of the experience being rendered for example, game genre, and device type of the SRC

A low confidence score may indicate that the estimation on the device is not adequate. In this case the server can try to re-estimate the pose and gaze prediction prior to rendering and encoding.

For foveated encoding this importance map is passed to the encoder to properly allocate bits for the encoding of the picture.

For foveated rendering, the SRS or the rendering engine in the SRS may similarly create an importance map to use to differentially allocate rendering resources for a frame.

When both foveated rendering and foveated encoding is used, the importance map used for rendering should take into account the importance map used for encoding and vice-versa.

A.3 Conformance Testing

No conformance testing procedures are defined in this version of the specification.

Annex B (normative): IDL Definition of Client API

The Split Rendering Client API is defined using the IDL syntax (according to ISO/IEC 19516) as follows:

```
interface SplitRenderer {
   readonly attribute SRState  state;

   attribute EventHandler onstatechange;
   attribute EventHandler onerror;
   attribute EventHandler onqualitychange;

   void SplitRenderer(in string application_id, in string aspId, in map
settings);

  void connect(in map settings, in List criteria);
   void disconnect(string reason);
   Metrics getMetrics(sequence<string> metrics);
};
```

Annex C (normative): Split Rendering Profiles

C.1 Pixel Streaming Profile

C.1.1 Introduction

This Annex defines split rendering profiles to define requirements for SRC and SRS for different scenarios. At this stage the following two profiles are defined:

- 2D Pixel Streaming Profile in clause C.2 to support split rendering to 2D screens, devices of type 3 in TS 26.119 [4].
- 3D Pixel Streaming Profile in clause C.3 to support split rendering to devices of type 1, 2, and 4 in TS 26.119 [4].

C.1.2 2D Pixel Streaming Profile

C.1.2.1 Introduction

This profile defines required capabilities for UE-based SRC functionalities as network-side SRS capabilities to support split rendering to 2D screens.

C.1.2.2 SRC Capabilities

C.1.2.2.1 Overview

Requirements for UE-based SRC functionalities for following functions are defined in this clause:

- Media Decoding
- Media Encoding
- Metadata Formats

The capabilities of the receiving UE are shared with the split rendering server prior to the start of the split rendering session.

C.1.2.2.2 Media Capabilities

The SRC shall support the media capabilities of a device type 3 as defined in TS 26.119 [4], clause 10.4.

C.1.2.2.3 Metadata Formats

XR-Pose-Cap 1: the SRC shall be able to retrieve one or more pose predictions for each view and for every frame to be rendered. The pose prediction shall be formatted according to clause 8.2.2.2.

XR-Pose-Cap 2: the SRC shall be able to retrieve and collect the user actions that occurred during an identified time interval. The action information shall be formatted according to clause 8.2.2.3.

C.1.2.3 SRS Capabilities

C.1.2.3.1 Overview

Requirements for network-based SRS functionalities for following functions are defined in this clause:

- Media Encoding
- Media Decoding
- Metadata Formats

The capabilities of the SRC are shared with the SRS prior to the start of the split rendering session.

C.1.2.3.2 Video encoding

The SRS shall at least be able to support the encoding of video that complies to the capabilities in clause 10.4.3 of TS 26.119.

C.1.2.3.3 Audio and Speech encoding

The SRS shall at least be able to support the encoding of audio that complies to the capabilities in clause 10.4.4 of TS 26.119.

C.1.2.3.4 Video decoding

The SRS has no requirements for the decoding of video streams.

C.1.2.3.5 Audio and Speech decoding

The SRS has no requirements for the decoding of audio or speech streams.

C.1.2.3.6 Metadata Formats

The SRS shall support the exchange of Pose and action information as defined in clause 8.3.2.

C.1.2.4 Profile identifier

The type **urn:3gpp:sr-mse:src:profile:2dpixelstreaming** shall be included in splitRenderingProfile parameter when the SRC signals SRS the Split Rendering Configuration [8.4.2.2].

C.1.3 3D Pixel Streaming Profile

C.1.3.1 Introduction

This profile defines required capabilities for UE-based SRC functionalities as network-side SRS capabilities to support MeCAR devices.

C.1.3.2 SRC Capabilities

C.1.3.2.1 Overview

Requirements for UE-based SRC functionalities for following functions are defined in this clause:

- Media Decoding
- Media Encoding

- Metadata Formats

The capabilities of the receiving UE are shared with the split rendering server prior to the start of the split rendering session.

C.1.3.2.2 Media Capabilities

The SRC shall support the media capabilities of a device type 1 as defined in TS 26.119 [4], clause 10.2.

If the device is a device type 2 as defined in TS 26.119 [4], clause 10.4, it shall also support the media capabilities of a device type 2 as defined in TS 26.119 [4], clause 10.3.

If the device is a device type 4 as defined in TS 26.119 [4], clause 10.5, it shall also support the media capabilities of a device type 2 as defined in TS 26.119 [4], clause 10.5.

C.1.3.2.3 Metadata Formats

XR-Pose-Cap 1: the SRC shall be able to retrieve one or more pose predictions for each view and for every frame to be rendered. The pose prediction shall be formatted according to clause 8.2.2.2.

XR-Pose-Cap 2: the SRC shall be able to retrieve and collect the user actions that occurred during an identified time interval. The action information shall be formatted according to clause 8.2.2.3.

C.1.3.3 SRS Capabilities

C.1.3.3.1 Overview

Requirements for network-based SRS functionalities for following functions are defined in this clause:

- Media Encoding
- Media Decoding
- Metadata Formats

The capabilities of the SRC are shared with the SRS prior to the start of the split rendering session.

C.1.3.3.2 Video encoding

The SRS shall at least be able to support the encoding of video that complies to the capabilities in clause 10.4.3 of TS26.119.

C.1.3.3.3 Audio and Speech encoding

The SRS shall at least be able to support the encoding of audio that complies to the capabilities in clause 10.4.4 of TS26.119.

C.1.3.3.4 Video decoding

The SRS has no requirements for the decoding of video streams.

C.1.3.3.5 Audio and Speech decoding

The SRS has no requirements for the decoding of audio or speech streams.

C.1.3.3.6 Metadata Formats

The SRS shall support the exchange of Pose and action information as defined in clause 8.3.2.

C.1.3.4 Profile identifier

The type **urn:3gpp:sr-mse:src:profile:3dpixelstreaming** shall be included in *splitRenderingProfile* parameter when the SRC signals SRS the Split Rendering Configuration [8.4.2.2].

C.1.4 Description of the Rendering Format for Pixel Streaming Profiles

C.1.4.1 General

In response to the Split Rendering Configuration message, the SRS shall reply with a description of the rendering format.

The rendering format description shall be a compliant glTF 2.0 [2] file. The file may include references to the buffer streams that contain the components of the rendered media.

Both SRS and SRC shall comply with the SD-Rendering-Ext1 capability as defined in TS 26.119 [4].

In addition, both SRS and SRC shall support for referencing WebRTC RTP streams and data channels as described in [3].

An SRC that complies with the 3D Pixel Streaming profile shall support the 3GPP_node_prerendered extension as defined in C.1.4.2.

C.1.4.2 3D Pixel Streaming Profile-specific glTF Extension

The 3GPP_node_prerendered extension is an extension at the node level to describe that the corresponding node is accessible as a prerendered content. The 3GPP_node_prerendered extension should be associated with the root node of the scene. It constitutes an alternative representation of the node and all its children. As such, if present, if the client decides to use the pre-rendered representation, it shall completely ignore the mesh description of the node and its children nodes.

The 3GPP_node_prerendered supports multiple 2D video textures and audio sources that correspond to the rendered views and audio content.

The semantics of the 3GPP_node_prerendered are provided by the following table:

Name	Type	Usage	Default	Description
visual	Object	O	N/A	An object that describes the rendered visual components of the content.
audio	Object	O	N/A	An object that describes the rendered audio components of the content.

The description of the visual object is provided in the following table:

Name	Type	Usage	Default	Description
visual_configuration	enum	O	VIEW_STEREO	An indication of the view configuration for the pre-rendered media. It can either be VIEW_MONO or VIEW_MONO.
Views	array(Object)	M		An array that describes the views of the prerendered content.
eye_visibility	enum	M		The visibility of the current view. This can take one of the following values: “EYE_LEFT”, EYE_RIGHT”, “EYE_BOTH”, or “EYE_NONE”.

Name	Type	Usage	Default	Description
				EYE_NONE is used for depth and transparency components.
composition_layers	array(number)	M		An array of accessors identifiers that each corresponds to a composition layer of the parent view.
composition_layer_type	array(enum)	M		For each of the composition layers of the parent view, this indicates the type of that composition layer. The values should be provided in the same order as the composition_layers. The allowed values are: "COMPOSITION_LAYER_PROJECTION", "COMPOSITION_LAYER_QUAD", "COMPOSITION_LAYER_EQUIRECTANGULAR", "COMPOSITION_LAYER_CUBEMAP", "COMPOSITION_LAYER_DEPTH", and "COMPOSITION_LAYER_OCCUPANCY".

The description of the audio object in the prerendered media extension is provided in the following table:

Name	Type	Usage	Default	Description
type	enum	O	AUDIO_STEREO	describes the format of the prerendered audio content. The type can take one of the following values: "AUDIO_MONO", "AUDIO_STEREO", and "AUDIO_HOA".
Components	array(number)	M		provides a list of the accessors that point to the media streams associated with rendered audio content.

The JSON scheme for the 3GPP_node_prerendered is as follows:

```
{
    "$schema" : "http://json-schema.org/draft-07/schema",
    "title" : "3GPP_node_rendered",
    "type" : "object",
    "description": "glTF extension to described pre-rendered content",
    "allOf": [ { "$ref": "glTFProperty.schema.json"} ],
    "properties" : {
        "visual": {
            "$ref": "3GPP_node_rendered.visual.schema.json",
            "description": "visual streamed buffers"
        },
        "audio": {
            "$ref": "3GPP_node_rendered.audio.schema.json",
            "description": "audio streamed buffers"
        },
        "extensions": {},
        "extras": {}
    },
```

```
    "required": ["visual"]
}
{
    "$schema" : "http://json-schema.org/draft-07/schema",
    "title" : "3GPP_node_rendered.visual",
    "type" : "object",
    "description": "Object representing the visual rendered media",
    "allOf": [ { "$ref": "glTFProperty.schema.json"} ],
    "properties" : {
        "view_configuration": {
            "type": "string",
            "description": "the view configuration used for the session",
            "gltf_detailedDescription": "the view configuration used for the
session",
            "enum": ["VIEW_MONO", "VIEW_STEREO"]
        },
        "views": {
            "type": "array",
            "description": "array of layer view objects",
            "gltf_detailedDescription": "",
            "items": {
                "$ref": "3GPP_node_rendered.visual.view.schema.json"
            },
            "minItems": 1
        },
        "extensions": {},
        "extras": {}
    },
    "required": ["views"]
}

{
    "$schema" : "http://json-schema.org/draft-07/schema",
    "title" : "3GPP_node_rendered.visual.view",
    "type" : "object",
    "description": "A representation of a rendered view",
    "allOf": [ { "$ref": "glTFProperty.schema.json"} ],
    "properties" : {
        "eye_visibility": {
            "type": "string",
            "description": "the visibility of the current view",
            "enum": ["EYE_LEFT", "EYE_RIGHT", "EYE_BOTH", "EYE_NONE"]
        },
        "composition_layers": {
            "type": "array",
            "description": "array of timed accessors that carry the streamed
buffers for each composition layer of the view",
            "items": {
                "type": "integer"
            },
```

```
            "minItems": 1
        },
        "composition_layer_type": {
            "type": "array",
            "items": {
                "type": "string",
                "description": "the type of composition layer in the array of
composition layers with the same array index",
                "gltf_detailedDescription": "the type of composition layer in the
array of composition layers with the same array index",
                "enum": ["COMPOSITION_LAYER_PROJECTION",
"COMPOSITION_LAYER_QUAD", "COMPOSITION_LAYER_EQUIRECTANGULAR",
"COMPOSITION_LAYER_CUBEMAP", "COMPOSITION_LAYER_DEPTH",
"COMPOSITION_LAYER_OCCUPANCY"]
            },
            "minItems": 1
        },
        "extensions": {},
        "extras": {}
    },
    "required": ["views"]
}

{
    "$schema" : "http://json-schema.org/draft-07/schema",
    "title" : "3GPP_node_rendered.audio",
    "type" : "object",
    "description": "Object representing the audio rendered media",
    "allOf": [ { "$ref": "glTFProperty.schema.json"} ],
    "properties" : {
        "type": {
            "type": "string",
            "description": "the type of the rendered audio",
            "gltf_detailedDescription": "the type of the rendered audio",
            "enum": ["AUDIO_MONO", "AUDIO_STEREO", "AUDIO_HOA"],
            "default": "AUDIO_STEREO"
        },
        "components": {
            "type": "array",
            "description": "array of timed accessors to audio component
buffers",
            "items": {
                "type": "integer"
            },
            "minItems": 1
        },
        "extensions": {},
        "extras": {}
    },
    "required": ["components"]
```

```
}
```

C.1.5 Profile Restrictions and Requirements

All Pixel Streaming profile are expected to be relocation intolerant and if using the 5G edge procedure shall set he *easRelocationRequirements* to “RELOCATION_INTOLERANT” in the tolerance field.

When the 2D Pixel Streaming profile is used, a policy template and a dynamic policy request may include the following QoS specifications, one for each of the components of the downlink streams:

- 1 QoS specification corresponding to the mono view.
- 1 QoS specification corresponding to one depth buffer stream associated with the mono view.
- 1 QoS specification corresponding to an occupancy/transparency buffer stream associated with the mono view.
- 1 QoS specification corresponding to an audio stream.

When the 3D Pixel Streaming profile is used, a policy template and a dynamic policy request may include the following QoS specifications, one for each of the components of the downlink streams:

- 2 QoS specifications corresponding to for left and right eye buffer streams.
- 2 QoS specifications corresponding to one depth buffer stream associated with the left and/or the right views.
- 2 QoS specification corresponding to an occupancy/transparency buffer stream associated with the left and/or the right views.
- 1 QoS specification corresponding to an audio stream.

C.2 Adaptive Split Rendering Profile

C.2.1 Introduction

This profile defines procedures and requirements for SRS and SRC to support split rendering features beyond a remote rendering paradigm.

Adaptive split rendering profile allows the SRC to render some objects of a scene locally and the rendering split can be adapted between the SRS and SRC during a session. The adaptation of the rendering split may be triggered either by the SRS or the SRC to maintain a consistent QoE of the SR session or to accommodate changes in operating conditions. The triggers may be, for example, channel conditions, SRC or SRS conditions or defined by the application provider.

To successfully render two parts of a scene separately in a split fashion, additional aspects of the rendering process need to be considered. Two basic requirements are maintaining a coherent state of the scene between the SRS and SRC and awareness of rendering split. Another requirement is seamless composition and display of the media rendered by the SRS and SRC into a frame to be displayed.

C.2.2 Procedures and Call Flows

For adaptive split rendering, the general procedures and call flows in clause 5.2 are followed with the following additions and modifications.

- The SRS and SRC should share a scene description. The implementation details may vary. The application provider may decide whether to provide identical scene descriptions to the SRS and the SRC or whether to provide a truncated version of the scene description to the SRC.

  Note: The Application Service Provider may provide the scene description resource to the SRS and SRC, for example, via M8 to SRC and via M2 to SRS.

- The SRS and SRC agree on an initial rendering split during session negotiation and the states to be synchronized, for example, in Step 5 of the procedure in clause 5.2.1.2.

- The initial rendering split and states to be synchronized are indicated in the SR configuration.

- In the rendering loop, exchange of split adaptation messages and state synchronization messages between the SRS and SRC is supported.

Figure C.2.2-1 illustrates a high level call flow set up and operation for a split rendering session which supports the adaptive split rendering profile.

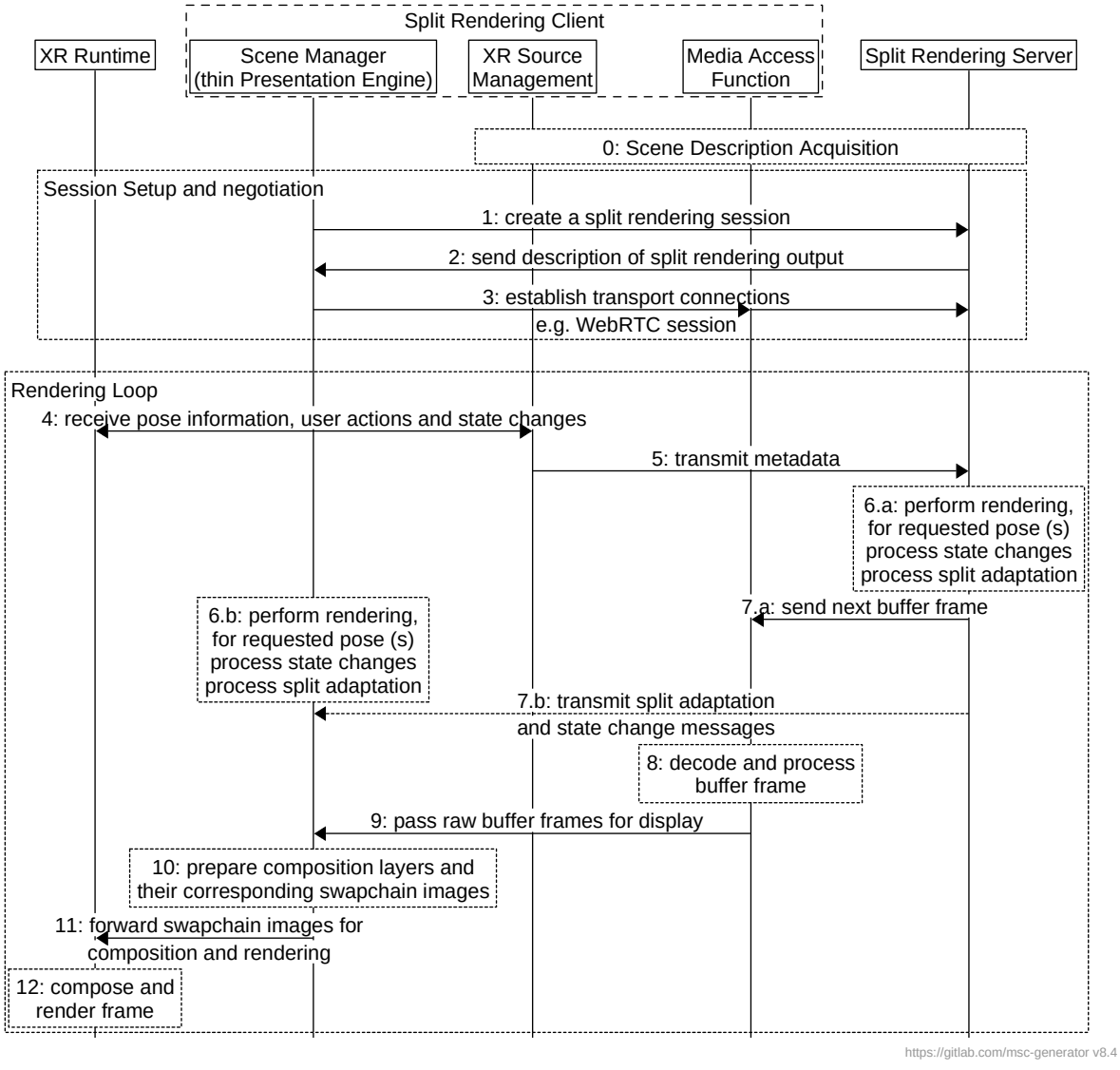

Figure C.2.2.-1: High level call flows for Adaptive Split Rendering Profile

The steps are:

0. In this optional step the SRC and the SRS acquire scene description of the scene to be rendered during the split rendering session. The actual implementation of delivery of the scene description by to the SRC and SRS is up to the application provider.

1. The Presentation Engine discovers the split rendering server and sets up a connection to it. It provides information about its rendering capabilities and the XR runtime configuration, e.g the OpenXR configuration

may be used for this purpose. States to be synchronized and the initial rendering split is negotiated during this step.

2. In response, the split rendering server creates a description of the split rendering output and the input it expects to receive from the UE.

3. The Presentation Engine requests the buffer streams from the MAF, which in turn establishes a connection to the split rendering server to stream pose and retrieve split rendering buffers.

4. The Source Manager retrieves pose and user input from the XR runtime and state changes in negotiated states and possible requests from the Scene Manager.

5. The Source Manager shares the pose predictions and user input actions , state changes and possible split adaptation messages with the split rendering server.

6. a. The split rendering server uses that information to, update states, render the frame and possibly update the split.

   b. The Scene Manager update states, renders a frame and possibly updates the split.

7. a. The rendered frame is encoded and streamed to the MAF.

   b. Possible split adaptation and state change messages are shared with the presentation engine,

8. The received media frames decoded and processed,

9. The raw buffer frames are passed to the Scene Manager, this includes the frames received from the SRS and the frames rendered locally by the PE,

10. The scene manager prepares composition layers and their corresponding swapchain images.

11. The swapchain images are forwarded to the XR runtime for composition and rendering,12. The frames are composed and displayed.

The final composition of a frame from media received from the SRS and locally rendered objects depends on the application logic. Implementation guidelines in C.2.7 provide a simple example.

C.2.3 Metadata Formats

C.2.3.1 Split Rendering Configuration Format

The configuration format defined in clause 8.4.2.2 with the additional fields defined below shall be used for split rendering configuration exchange in adaptive split rendering profile.

renderingSplit	Object	1..1	A object identifying objects to be rendered and where they are to be rendered (SRS or SRC), for example, as a dictionary with keys “SRS” and “SRC” and lists of object indices from a scene description or a scene graph
synchronizedStatesInit	Object	1..1	A object identifying states to be synchronized between the SRS and SRC and their initial state
states	Object	1..1	A list of state identifiers, their current values
state	String/number	1..n	Identifier of a state
initVal	String	1..n	Initial value of the state
stateVals	Array	1..1	An array of values possible for the state

These renderingSplit object shall be present as part of the extraConfigurations Object as defined in clause 8.4.2.2 for extensibility of split rendering configuration format.

C.2.3.2 Split Adaptation Message Format

During a split rendering session, the operating environment of the split rendering server, the split rendering client or the network conditions may change. Consequently, the rendering split may need to be adapted to deliver a consistent QoE. When adaptive split rendering is enabled, the SRS or SRC shall request a new rendering split by sending a message of the type "urn:3gpp:split-rendering:v1:asrp:sr-split". The message shall be conformant to the metadata message format specified in clause 8.5.1. The same message type shall be used to acknowledge, accept or reject the request by the receiver, with the message subtype identifying whether it is a request, acceptance, acknowledgement or rejection. The message shall follow the format in Table C.2.3.1-1.

Table C.2.3.2-1 Message format for split adaptation messages

Name	Type	Cardinality	Description
id	string	1..1	A unique identifier of the message in the scope of the data channel session.
type	string	1..1	urn:3gpp:split-rendering:v1:asrp:sr-split
message	Object	1..1	Message content
subtype	string	1..1	An identifier of the subtype of the message, it may be a request (REQ) for new split or acknowledgement (ACK), acceptance (OK) or rejection of a request (NOK).
renderingSplitId	string	1..1	An identifier of the rendering split unique within the scope of the SR session
renderingSplit	Object	0..1	A object identifying objects to be rendered and where they are to be rendered (SRS or SRC). The message shall be a dictionary object . with keys "SRS" and "SRC", and values corresponding to a key shall be a list of named nodes from the scene description being rendered in the SR session. The keys shall indicate where the objects named in the corresponding value list are rendered.

Split adaptation messages indicating acceptance, acknowledgment or rejection of a split adaptation request may not include the renderingSplit Object.

C.2.3.3 State Synchronization Message Format

During a split rendering session, various states associated with the scene being rendered may transition. Depending on the nature of the application being executed, a transition may occur at the SRS, at the SRC or at both the SRS and SRC. For the application execution to be consistent, some state transitions need to be synchronized between the SRS and SRC. The SRC and SRS may agree on which states to synchronize during session setup. To synchronize state transitions during a split rendering session the SRS and SRC shall exchange messages of the type "urn:3gpp:split-rendering:v1:asrp:sr-state" . The same message type shall be used to send a state synchronization update, acknowledge a state synchronization update or simultaneously send and acknowledge a state synchronization update. The state synchronization update messages shall be conformant with the meta-data message format defined in clause 8.5.1 and the message content shall be formatted as shown in Table C.2.3.2-1.

Table C.2.3.2-1 Message format for state synchronization messages

Name	Type	Cardinality	Description
id	string	1..1	A unique identifier of the message in the scope of the data channel session.
type	string	1..1	**urn:3gpp:split-rendering:v1:sr-state**
message	Object	1..1	Message content
subtype	string	1..n	An identifier of the subtype of the message, it may be a state synchronization update (SYNC), acknowledgment (ACK) or both (SYNC_ACK)
syncUpdateId	string	1..1	An identifier of the synchronization update unique within the scope of the SR session
synchronizedStates	Object	1..1	An object identifying states that are synchronized between the SRS and SRC and their current state. Only states that have transitioned may be exchanged
states	Object	1..1	A list of state identifiers, their current values and last change time
identifier	String/number	1..n	Identifier of a state
val	Object/String/number	1..n	Value of the state
lastChangeTime	number	1..1	The timestamp of the last change in state

Split adaptation messages indicating an acknowledgment of a state update may not include the synchronizedStates Object.

C.2.4 SRC Capabilities

The adaptive split rendering profile may be used in monoscopic mode or stereoscopic mode. In monoscopic mode, the SRC receives video corresponding to a single view. This mode supports split rendering to 2D screens, devices of type 3 in TS 26.119 .

In stereoscopic mode, the SRC receives video corresponding to two views, one for each eye. This mode supports split rendering to stereoscopic screens, devices of type 1,2,4 in TS 26.119.

C.2.4.1 Media Capabilities

If adaptive split rendering profile is used in monoscopic mode, the SRC shall support the media capabilities of a device type 3 as defined in TS 26.119 [4], clause 10.4, and referenced in Annex C.1.2.2.2 .

If adaptive split rendering profile is used in stereoscopic mode, the SRC shall support the media capabilities for device type 1 as defined in TS 26.119 [4], clauses 10.2, and referenced in Annex C.1.3.2.2

If the device is a device type 2 as defined in TS 26.119 [4], clause 10.4, it shall also support the media capabilities of a device type 2 as defined in TS 26.119 [4], clause 10.3, and referenced in Annex C.1.3.2.2

If the device is a device type 4 as defined in TS 26.119 [4], clause 10.5, it shall also support the media capabilities of a device type 4 as defined in TS 26.119 [4], clause 10.5, and as referenced in Annex C.1.3.2.2

C.2.4.2 Metadata Formats

XR-Pose-Cap 1: the SRC shall be able to retrieve one or more pose predictions for each view and for every frame to be rendered. The pose prediction shall be formatted according to clause 8.3.2.2.

XR-Pose-Cap 2: the SRC shall be able to retrieve and collect the user actions that occurred during an identified time interval. The action information shall be formatted according to clause 8.3.2.3.

XR-ObjId-Cap 1: the SRC shall be able to receive, retrieve and collect identifiers of objects in a scene being rendered by the SRC in a split rendering session during an identified time interval. The state information shall be formatted according to clause C.2.3.2

XR-ObjState-Cap 1: the SRC shall be able to receive, retrieve and collect state changes in identified objects in a scene being rendered in an split rendering session during an identified time interval. The state information shall be formatted according to clause C.2.3.3

C.2.4.3 Rendering format description

The SRC and SRS shall comply with rendering format description in annex C.4.1

If adaptive split rendering profile is used for stereoscopic use cases, the SRC shall support the 3GPP_node_prerendered extension defined in C.4.2, However, the extension may be used on non-root nodes.

C.2.4.4 Scene Processing and Rendering Capabilities

The SRC shall have the following minimum scene processing capabilities:

- the ***SD-Rendering-gltf-core*** scene processing capabilities defined in clause 9.2 of TS 26.119.

SD-Rendering-gltf-core enables basic compatibility of an SRC with the adaptive split rendering profile for simple use cases, where the SRC does minimal local rendering and adaptability of rendering split is minimal. An example of such a limited scenario may be a cloud gaming use case where the application provider isolates a small subgraph of the complex game scene to be rendered by the SRC and shares the subgraph with the SRC. The subgraph may contain only the assets (mesh and textures) related to a user's character and controller to allow the SRC to render these objects locally to mask motion to photon to render latency. More advanced use cases of adaptive split rendering place higher scene processing capabilities on the SRC.

The SRC should have the following scene processing capabilities:

- the ***SD-Rendering-gltf-Ext1*** scene processing capabilities defined in clause 9.2 of TS 26.119.

In addition to the above specified scene processing capabilities, depending on the device type, the SRC shall have scene capabilities defined for each device type in clause 10 of TS 26.119.

C.2.5 SRS Capabilities

The SRS capabilities to support adaptive split rendering profile are described in the sub-clauses below.

C.2.5.1 Media Capabilities

The media capabilities of the SRS are defined in relation to the media capabilities of the SRCs it is expected to serve. Therefore, the encoding capabilities of an SRS should match the decoding capabilities of the SRC.

If adaptive split rendering profile is used in monoscopic code, the SRS shall have capabilities defined in clause C.1.2.3

If adaptive split rendering profile is used in stereoscopic mode, the SRS shall have capabilities defined in clause C.1.3.3.

C.2.5.2 Metadata Capabilities

The SRS shall support the metadata formats for pose and action defined in Clause 8.3.2. In addition, the SRS shall support the metadata formats defined in Annex C.2.3, and complement the metadata capabilities defined in Annex C.2.4.2. This shall include the ability to receive and process messages corresponding to metadata capabilities defined in Annex C.2.4.2 and formatted according to clause 8.3.2 and C.2.3.

C.2.5.3 Scene Processing and Rendering Capabilities

SRS shall have the **SD-Rendering-gltf-Ext1** scene processing capabilities.

Additionally, depending on the device type of the SRC participating in a split rendering session, the SRS should support the required and recommended scene processing capabilities defined in TS 26.119 in clause 10.3.5 for device type 2, clause 10.4.5 for device type 3, and 10.5.5 for device type 4.

C.2.6 Profile identifiers

If the adaptive split rendering profile is used in monoscopic modethe type **urn:3gpp:sr-mse:src:profile:asr2dpixelstreaming** shall be included in *splitRenderingProfile* parameter when the SRC signals SRS the Split Rendering Configuration [8.4.2.2].

If the adaptive split rendering profile is used in in stereoscopic mode the type **urn:3gpp:sr-mse:src:profile:asr3dpixelstreaming** shall be included in *splitRenderingProfile* parameter when the SRC signals SRS the Split Rendering Configuration [8.4.2.2].

C.2.7 Extension to Client API Functions

The SRC should perform adaptive split management which may be based on metrics reports of an ongoing split rendering session, scene being rendered and UE operating conditions. For adaptive split rendering, the SRC exposes functions to load and update scene description resources. The SRC may also expose functions to an application to allow application developers to deploy custom logic for split management.

Method	**Parameters**		**State after Success**	**Description**
	in	**out**		
setScene()	- srSessionId -scene description resource	-status	N/A	The application requests the SRC to load a scene description resource for rendering in the split rendering session.
updateScene()	- srSessionId -scene description resource	-status	N/A	The application request the SRC to update a scene description resource being rendered in the split rendering session.

SRC may optionally expose the function below to the application to allow application developers to deploy custom split management logic.

updateSplit()	- srSessionId -rendering split	-status -rendering split	N/A	The application requests or queries the SRC for a new rendering split or the current rendering split in use

The parameters used are defined below:

- srSessionId: as defined in Clause 9.2

- scene description resource: A scene description resource compliant with capabilities specified in clause C.2.4.4. The scene description resource may be a subset of the scene description resource being rendered by the SRS. It is assumed that the application provider makes the scene description resource available to the application, for example, via M8.
- status: indicates whether the call was successful (OK) or not successful (FAIL)
- rendering split: A pointer to a renderingSplit object defined in C.2.3.

C.2.8 Implementation Guidelines for Adaptive Split Rendering

C.2.8.1 General

ASR profile may be used with any of the pixel streaming profiles defined in Annex C. As such implementation guidelines for pixel streaming profiles may be applicable. Additional guidelines are provided below, and where applicable, differences from pixel stream profile guidelines are highlighted.
The ASR profile allows SRS and SRC both to render objects for a given display frame. This functionality may be leveraged by application developers to develop SR applications which are more responsive to user interaction by leveraging local rendering for interactive objects. However, caution has to be exercised in choosing which objects to render locally and which objects to render remotely in a given 3D scene. The division should be such that composition of the locally rendered and remotely rendered frames into a display frame is visually as seamless as possible for the user.
The logic on how to choose objects for local or remote rendering and how to compose the frame from locally and remotely rendered frames is left to the Application provider and application developers. Below we provide guidelines based on a simple use case to illustrate how to split objects for rendering and how to compose the final frame.

C.2.8.2 Guidelines for Rendering Split and Composition

Adaptive split rendering can be used based on the type of AR object. In this case it is most suited to AR objects that have low prediction accuracy, i.e., pose prediction and reprojection techniques are not enough to make up for the motion-to-render latency when using remote rendering. For example,

- Interactive objects that react to user actions, pose, eye gaze, stimuli in the environment, etc.
- Objects with high reflectivity, especially in the presence of motion in the environment, such as, moving objects, changing light conditions, etc.
- Transparent objects

In most cases, interactive objects can benefit from being rendered locally if the device has the capability to do so. However, to limit the number of objects rendered by the device, an application may choose to render only those interactive objects on the device that are near the user, e.g., they are located within a radius surrounding the device. Since the user is unlikely to interact with objects that are farther, i.e., outside this radius, these can still be rendered remotely. The application may also choose to render only interactive objects that have high level of interactivity on the client and render objects with predictable motion and slow responses on the server.

Since, the level of interaction and distance from the user can change during the lifetime of the session, the rendering is appropriately adapted.

In most cases, reflective objects may need to be rendered remotely as they require higher processing. If reflective objects are rendered at the client, then for convincing reflection effects, it is important for the server to provide an environment map to the client so the client can use it for shading its local objects. In practice, the client needs a (low-resolution) 360-degree cube map of the viewer's entire surroundings, with local objects omitted. The client may still need to locally augment this environment map with local objects in case they are prominently featured in reflections, for instance if they are very large or have bright light sources.

Annex X (informative): Change history

Change history							
Date	Meeting	TDoc	CR	Rev	Cat	Subject/Comment	New version
04-2023	123-e	S4-230726				Improvements and Corrections to edge and dynamic policy procedures in SR	
05-2023	124	S4-121075				General updates to TS26.565	
05-2023	124	S4-121004				SR Rendering API	
05-2023	124	S4-231005				Pixel Streaming Media Profile	
05-2023	124	S4-231003				pCR on signaling for SR session control	
05-2023	124	S4-230925				On SR configuration API and view configuration	
08-2023	125	S4-231449				[SR_MSE] Transport protocols	
08-2023	125	S4-231518				[SR_MSE] Rendering optimization	
08-2023	125	S4-231432				[SR_MSE] Updates to Media Capabilities	
08-2023	125	S4-231324				Split rendering Metrics	
08-2023	125	S4-231434				Editorial corrections on SR MSE architectures	
11-2023	126	S4-231909				Editor’s updates	0.7.0
11-2023	126	S4-231911				Added pose interval to configuration	0.7.0
11-2023	126	S4-231912				Added signaling of SR profile in configuration	0.7.0
11-2023	126	S4-231914				Clarified session setup and configuration	0.7.0
11-2023	126	S4-231796				Added protocol stack	0.7.0
11-2023	126	S4-232007				Timing information in QoE metrics	0.7.0
11-2023	126	S4-231800				Made fov optional in pose format	0.7.0
11-2023	126	S4-231802				Defined output signaling format for pixel streaming	0.7.0
11-2023	126	S4-232011				Updated media profiles for pixel streaming profile	0.7.1
12-2023	SA#102	SP-231306				Version 1.0.0 created by MCC	1.0.0
02-2024	127	S4-240404				TS cleanup	1.1.0
02-2024	127	S4-240135				Profile identifiers	1.1.0
02-2024	127	S4-240400				Pre-requisites for Split Rendering	1.1.0
02-2024	127	S4-240405				Device Type	1.1.0
02-2024	127	S4-240198				Editorial corrections	1.1.0
02-2024	127	S4-240422				QoE metrics timing information format	1.1.0
04-2024	127-e	S4-240786				Corrections and Guidelines for TS26.565	1.2.0
04-2024	127-e	S4-240810				[SR_MSE] pCR on Adaptive Split Rendering Profile	1.2.0
04-2024	127-e	S4-240581				[SR_MSE]pCR Editorial Corrections	1.2.0
04-2024	127-e	S4-240582				[SR_MSE]pCR ASR Profile Implementation Guidelines	1.2.0
05-2024	128	S4-241006				Clarification on RTC-6 interface in SR_MSE architecture	1.3.0
05-2024	128	S4-241140				Client API in Split Rendering	1.3.0
05-2024	128	S4-241142				Editorial corrections in TS 26.565	1.3.0
05-2024	128	S4-241246				ASR profile client API	1.3.0
05-2024	128	S4-241262				QoE metrics reporting for Split Rendering Client	1.3.0
06-2024						Version 2.0.0 created by MCC to be sent to TSG for approval	2.0.0
06-2024						Version 18.0.0 created by MCC upon approval in TSG	18.0.0

History

Document history		
V18.0.0	July 2024	Publication