**3GPP SA4 #123-e S4-230729**

**17-21 April 2023**

|  |
| --- |
| *CR-Form-v12.0* |
| **PSEUDO CHANGE REQUEST** |
|  |
|  | **26**.**119** | **CR** | pseudo | **rev** | **-** | **Current version:** | **0.1.0** |  |
|  |
| *For* [***HE******LP***](http://www.3gpp.org/3G_Specs/CRs.htm#_blank)*on using this form: comprehensive instructions can be found at* [*http://www.3gpp.org/Change-Requests*](http://www.3gpp.org/Change-Requests)*.* |
|  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Proposed change affects:*** | UICC apps |  | ME |  | Radio Access Network |  | Core Network |  |

|  |
| --- |
|  |
| ***Title:***  | **[MeCAR] Merged Draft Specification** |
|  |  |
| ***Source to WG:*** | Qualcomm Incorporated, Xiaomi, Tencent |
| ***Source to TSG:*** |  |
|  |  |
| ***Work item code:*** | MeCAR |  | ***Date:*** | 17/04/2023 |
|  |  |  |  |  |
| ***Category:*** |  |  | ***Release:*** | Rel-18  |
|  | *Use one of the following categories:****F*** *(correction)****A*** *(mirror corresponding to a change in an earlier release)****B*** *(addition of feature),* ***C*** *(functional modification of feature)****D*** *(editorial modification)*Detailed explanations of the above categories canbe found in 3GPP [TR 21.900](http://www.3gpp.org/ftp/Specs/html-info/21900.htm). | *Use one of the following releases:Rel-10 (Release 10)Rel-11 (Release 11)Rel-12 (Release 12)**Rel-13 (Release 13)Rel-14 (Release 14)Rel-15 (Release 15)Rel-16 (Release 16)* *Rel-17 (Release 17)* *Rel-18 (Release 18)* |
|  |  |
| ***Reason for change:*** |  |
|  |  |
| ***Summary of change:*** | Merge of proposals on how to structure the Draft TS, updates on the Device architecture and initial transfer of PD material |
|  |  |
| ***Consequences if not approved:*** |  |
|  |  |
| ***Clauses affected:*** |  |
|  |  |
|  | **Y** | **N** |  |  |
| ***Other specs*** |  |  |  Other core specifications  | TS/TR ... CR ...  |
| ***affected:*** |  |  |  Test specifications | TS/TR ... CR ...  |
| ***(show related CRs)*** |  |  |  O&M Specifications | TS/TR ... CR ...  |
|  |  |
| ***Other comments:*** |  |
|  |  |
| ***This CR's revision history:*** |  |

|  |
| --- |
| 3GPP TS 26.119 V0.2.0 (2023-04) |
| Technical Specification |
| 3rd Generation Partnership Project;Technical Specification Group Services and System Aspects;Media Capabilities for Augmented Reality(Release 18) |
|   |
|  |  |
|  |
| The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP.The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented.This Specification is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification.Specifications and Reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices. |

|  |
| --- |
|  |
| ***3GPP***Postal address3GPP support office address650 Route des Lucioles - Sophia AntipolisValbonne - FRANCETel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16Internethttp://www.3gpp.org |
| ***Copyright Notification***No part may be reproduced except as authorized by written permission.The copyright and the foregoing restriction extend to reproduction in all media.© 2021, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).All rights reserved.UMTS™ is a Trade Mark of ETSI registered for the benefit of its members3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational PartnersLTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational PartnersGSM® and the GSM logo are registered and owned by the GSM Association |

Contents

Foreword 6

Introduction 7

1 Scope 8

2 References 8

3 Definitions of terms, symbols and abbreviations 8

3.1 Terms 8

3.2 Symbols 9

3.3 Abbreviations 9

4 Prerequisites 10

4.1 Hypothetical XR Runtime and related concepts 10

4.1.1 Overview 10

4.1.2 XR Session and Rendering Loop 11

4.1.3 XR Runtime Capabilities 13

4.2 Media Pipelines and Rendering Loop 15

4.2.1 General 15

4.2.2 Basic Media Pipeline 16

4.2.3 Advanced Media Pipelines 16

4.2.4 Rendering capabilities 16

4.3 Application and Service Provider view 16

4.4 Structure of the specification 16

5 Device reference architecture and interfaces 16

5.1 Architecture 16

5.2 Description of the functional blocks 17

5.3 Interfaces and APIs 17

6 General and systems functions and capabilities 18

6.1 Metadata formats 18

6.1.1 General 18

6.1.2 Pose Prediction Format 18

6.1.3 Action Format 20

7 Visual functions and capabilities 20

7.1 Decoding Capabilities 20

7.1.1 Video Decoding 20

7.1.2 Video decoding interface 21

7.2 Encoding Capabilities 21

7.2.1 Video Encoding 21

7.3 Scene Processing Capabilities 21

7.4 Capability exchange 21

8 Audio functions and capabilities 21

9 QoE Metrics 21

10 Device Types and Media Profiles 21

10.1 Introduction 21

10.2 Device Type 1: Thin AR Glasses 21

10.3 Device Type 2: AR Glasses 22

10.3 Device Type 3: XR Phone 22

10.4 Device Type 4: XR HMD 23

Annex A (informative/normative): KPIs for AR/MR 24

A.1 Introduction 24

Annex B (informative): Usage of OpenXR [and WebXR] as XR Runtime 24

B.1 Introduction 24

B.2 Capability Mapping to OpenXR 24

4.1.4 XR Views and Rendering Loop 24

[B.3 Capability Mapping to WebXR] 26

Annex <X> (informative): Change history 27

# 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 clause is optional. If it exists, it shall be the second unnumbered clause.

# 1 Scope

The present document …

# 2 References

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

- References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

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

[2] 3GPP TR 26.928: "Extended Reality (XR) in 5G".

[3] 3GPP TR 26.998: "Support of 5G glass-type Augmented Reality / Mixed Reality (AR/MR) devices".

[4] 3GPP TR 26.857: "5G Media Service Enablers".

[5] Khronos, "The OpenXR Specification", https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html…

# 3 Definitions of terms, symbols and abbreviations

## 3.1 Terms

For the purposes of the present document, the terms given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

**Frame of Reference**: an abstract coordinate system whose origin, orientation, and scale are specified by a set of reference points

**Reference Points**: geometric points whose position is identified both mathematically and physically.

**XR Application:** application running on an XR Device which offers an XR experience based on an XR Runtime

**XR Device:** a device capable of offering an XR experience.

**XR Runtime**: Set of functions provided by the XR Device to the XR Application in order to create XR experiences.

**XR Runtime API**: the API to communicate with an XR Runtime

**XR Session**: an application’s intention to present XR content to the user.

**XR View:** a rendered view of the scene generated by the XR Application and passed on to the XR Runtime during a running XR Session

## 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 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

AR Augmented Reality

MR Mixed Reality

VR Virtual Reality

XR eXtended Reality

# 4 Prerequisites

## 4.1 Hypothetical XR Runtime and related concepts

### 4.1.1 Overview

Extended Reality (XR) refers to a continuum of real-and-virtual combined environments generated by computers through human-machine interaction. XR encompasses technologies associated with virtual reality (VR), augmented reality (AR) and mixed reality (MR). A detailed overview of definitions, concepts and background on XR and AR is provided in TR 26.928 [2] and TR 26.998 [3], respectively.

The clause 4 documents the core assumptions for a device capable of offering an XR experience. In the context of this document, such devices will be referred to an XR Device. An XR Device is assumed to have one or several displays, speakers, sensors, cameras, microphones, actuators, controllers and/or other peripherals that allow to create XR experiences, i.e. experiences for which the user interacts with the content presented in virtual world and/or augmented to the real-world. Example of XR Devices are for example AR Glasses, a VR/MR Head-Mounted Display (HMD) or a regular smartphone.

An application which offers an XR experience by making use of the hardware capabilities, including media capabilities, of the XR Device it runs on as well as the network connectivity to retrieve the asset being used by the application is referred to as an XR Application. In the context of this specification, it is primarily assumed that access to the network is provided by 5G System functionalities.

To enable XR experiences, the hardware on an XR Device typically offers a set of functions to perform commonly required XR operations. These operations include, but are not limited to:

- accessing controller/peripheral state,

- getting current and/or predicted tracking positions and pose information of the user,

- receiving pre-rendered views of the scene for final presentation to the user, taking into account the latest user position and pose. Adaptation to the latest user position and pose is also referred to as warping.

The set of functions provided by the XR Device to the XR Application in order to create XR experiences is defined as XR Runtime. The functions are accessible to the XR Application via an API exposed by the XR Runtime referred to as the XR Runtime Application Programming Interface (API). The XR Runtime typically handles functionalities such as composition, peripheral management, tracking, Spatial Localization and Mapping (SLAM), capturing and audio-related functions. Further, it is assumed that the hardware and software capabilities of the XR Device are accessible through well-defined device APIs, and in particular the media capabilities are accessible through media APIs.

In the remainder of the specification, the XR prefix with Runtime or Application or other defined XR-prefixed terms may be omitted for better readability.

An overview of an XR Device logical components is shown in Figure 4.1.1-1.



Figure 4.1.1-1 Logical components of an XR Device

The primary scope of this specification is the definition of a minimum amount of media capabilities that an XR Application can rely when deployed on a certain type of XR Device. Media capabilities include, but are not limited to, media encoders and decoders, parsing and writing of media encapsulation format, security functions, synchronization information, spatial alignment information, metadata formats, graphics capabilities, etc.

The logic and behaviour of the XR Application is not specified in this specification, but the XR Application may be a 3GPP-based service or a third-party service. The media capabilities may also be referenced as part of a Media Session Enabler as defined in TR 26.857 [4].

This specification relies on an hypothetical XR Runtime and its API in order to define the media capabilities. This way, different implementation of XR runtimes may be compatible with this specification. However, for the purpose of developing this specification, a subset of the expected functionalities of the XR Runtime has been aligned with what is offered by the core Khronos’ OpenXR specification [5]. The OpenXR specification has been used as a reference for defining the XR Runtime API and functionalities which guarantees at least the compatibility of the present specification with a XR Runtime as defined in the OpenXR specification. Although not required, the readers of this specification are encouraged to familiarize themselves with the OpenXR specification and concepts which gives an in-depth understanding of the internal mechanics of an XR Device. Lastly, a mapping of general functionalities to OpenXR is provided in Annex B.

### 4.1.2 XR Session and Rendering Loop

At startup, the XR Application attempts to create an XR Session via the XR Runtime API using the available resources on the XR Device. Upon success, the XR Runtime begins the life-cycle of the XR Session which goes through different states. Those states are meant to synchronise the rendering operations controlled the XR Application with the display operations controlled by the XR Runtime. The rendering loop is thus a joint task executed by the XR Runtime and the XR Application synchronised via the states of the XR Session.

The XR application is responsible of presenting a rendered view of the scene to the user. For this purpose, the XR application generates XR Views which are passed to the XR Runtime at every iteration of the rendering loop. The XR Views are generated for a meaningful set of one or more poses in the scene for which the XR application can render images. A XR View configured as a primary view is intended to be presented to the viewer interacting with the XR application. Additional XR Views may be provided, for example views from poses which are intended for spectators. The focus in the remainder is on the primary view configuration for the interacting viewer.

A typical head-mounted XR System has a stereoscopic view configuration, i.e. two views, while a handheld XR Device has a monoscopic view configuration, i.e. a single view. Other view configurations may exist. The application selects its primary view configuration type at the start of the session, and that configuration remains constant for the lifetime of the session.

A XR View comprises one more composition layers associated with an image buffer. Those composition layers are drawn in a specified order, with the 0th layer drawn first. Layers are drawn with a “painter’s algorithm,” with each successive layer potentially overwriting the destination layers whether or not the new layers are virtually closer to the viewer. Composition layers are subject to blending with other layers. Blending of layers can be controlled by the alpha channel information present in the image buffer of each layer. In addition, the image buffer of the layer may be limited by a maximum width and a maximum height when rendering them such that they fit into the capabilities of the swapchains.

In addition to layers containing visual data, an XR View may be complemented with a layer provided depth information of the view. This added information may help the XR Runtime to do pose correction when generating the final display buffer.

In order to allow the application in rendering XR Views, the XR Runtime provides the viewer pose and projection parameters that are needed to predict and render the different XR Views. The viewer pose and projection parameters for are provided for a particular display time in the near future. The XR Runtime typically allows to be repeatedly called for prediction updates of the pose, which may not necessarily return the same result for the same target display time.

Instead, the prediction gets increasingly accurate as the function is called closer to the given time for which a prediction is made. This allows an application to prepare the predicted views early enough to account for the amount of latency in the rendering while at the same time minimising the prediction error when pre-rendering the views.

In addition, XR applications communicate with input devices in order to collect actions. Actions are created at initialization time and later used to request input device state, create action spaces, or control haptic events. Input action handles represent ‘actions’ that the application is interested in obtaining the state of, not direct input device hardware.

[

Pose: <https://registry>.khronos.org/OpenXR/specs/1.0/html/xrspec.html#input

Media Data: Camera, Microphone

- Assumption:

- Do we include capturing in the first version? Minimize the functionalities (For example a Time-of-Flight or a avatar support camera output)

- We do less well-defined and extend later in future releases.]



Figure 4.1.4-1 Rendering loop for visual data

[For audio rendering, the following processes may be assumed:

1) An interface to the XR runtime is available hand over raw audio buffers to determine how the XR application would access a device’s audio capabilities. In audio, typically the term buffer queues is used instead of swap chains but they may be viewed equivalent to visual swap chains.

2) In addition to the functionalities from such buffer queues, different types of audio signals may be provided, and additional/alternative processing steps may be carried out. Audio signals (i.e. the combination of metadata and buffer queues) may be

a. non-diegetic, i.e. they are not rendered to the pose.

b. Provide a projected audio format that can be processed by the XR runtime to adjust to the user pose

c. a mixture of such signals that are jointly presented, equivalent to the composition done for the visual

3) Details are tbd, also in relation to TS 26.119. clause 4.5 and Annex B

The composition and display in the run-time also includes the audio-visual synchronization.

Composition layers submitted by the application include an XrSpace for the runtime to use to position that layer over time. Composition layers whose XrSpace is relative to the VIEW reference space are implicitly “head-locked”, even if they may not be “display-locked” for non-head-mounted form factors. The application typically uses multiple media types and may use multiple composition layers per media type. For visual media, composition layers represent independent images that the compositor blends together to produce the final displayed image. Composition layers may be

- “head-tracked” if desired to be corrected to the latest user pose

- “head-locked” if the content is not adapted to the latest user pose in an HMD.

- “display-locked” for not following the latest user pose.

- “body-locked” for not following the latest user pose.

In audio, such composition layers are referred to as diegetic (“head-tracked”) or non-diegetic (“head-locked”).]

### 4.1.3 XR Runtime Capabilities

Table 4.1.5-1 provides a summary of relevant capabilities for XR Runtimes. This table does not prescribe support for any specific capabilities, this is addressed for each device type individually. A mapping of these high-level capabilities to OpenXR is provided in Annex A.

Table 4.1.5-1 XR Runtime Capabilities

|  |  |  |  |
| --- | --- | --- | --- |
| Capability | Description and Reference | Parameters | OpenXR (will be moved to Annex) |
| XR System Properties | An application can query the XR Runtime to retrieve information about the system such as a system identifier, graphics properties or tracking properties. | System identifierTracking PropertiesGraphics Properties | xrGetSystemProperties |
| XR System Graphics Properties | Information on the graphics capabilities, namely the maximum image pixel height and width of the swapchain as well as the maximum number of composition layers | maxSwapchainImageHeight maxSwapchainImageWidth maxLayerCount  | xrSystemGraphicsPropertiesminMaxLayerCount = 16 |
| XR System Tracking Properties | Information on the tracking capabilities, namely support of orientation and position tracking. | orientationTrackingpositionTracking | XrSystemTrackingProperties |
| Blend Mode | The supported blend modes of the XR System, see clause 4.1.4 | Opaque, additive, alpha\_blend | XrEnvironmentBlendMode |
| Supported view configuration types | Supported primary view configurations by the XR System | Mono, Stereo, others | xrEnumerateViewConfigurations[xrViewConfigurationType](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22XrViewConfigurationType)  |
| View Configuration Properties | specifies properties related to rendering of an individual view within a view configuration | Recommended and maximum height/width and swapchain sample count | [XrViewConfigurationView](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22XrViewConfigurationView) |
| Reference Space Type | XR Runtimes implement different reference spaces as described in clause 4.1.3 | View, Local, Stage, unbounded, user-defined | [xrEnumerateReferenceSpaces](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22xrEnumerateReferenceSpaces) |
| Spatial Range Boundaries | XR systems may have limited real world spatial ranges in which users can freely move around while remaining tracked | dimensions of an axis-aligned bounding box | [xrGetReferenceSpaceBoundsRect](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22xrGetReferenceSpaceBoundsRect) |
| Swapchain Formats | Swapchain image format support by the runtime | For example R8G8B8A8 | [xrEnumerateSwapchainFormats](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22xrEnumerateSwapchainFormats) |
| Swapchain Images | number of images allocated to swapchain | For example 1 or 2 | xrEnumerateSwapchainImages |
| Projection Layer Type | Provides the supported layer type that is used in the projections for the layer | Projection Composition Layer: represents planar projected images, one rendered for each eye using a perspective projection.Quad Composition Layer: is useful for rendering user interface elements or 2D content on a planar area in the world. Cylinder Composition Layer: the XR runtime maps a texture stemming from a swapchain onto the inside of a cylinder section. Cube Composition Layer: consists of a cube map with 6 views to be rendered by the application.Equirectangular Composition Layer: consists of an equirectangular image that is mapped onto the inside of a sphere in the world.Depth Composition Layer: provides an extra composition layer to allow applications to submit depth maps to assist with the pose correction of projected images of a project layer. | XrStructureType |
| Frame rate |  |  |  |
| ACTIONS |  |  |  |

[Add a table of capabilities of the XR Runtime and what is expected to available and what is optional needs to be queried.

Basic concept of specification:

- Capability query

- [Editor’s note: Description of the pipelines, sensors, AR runtime, decoders… identify for what entities capabilities are defined]

Collected Requirements]

## 4.2 Media Pipelines and Rendering Loop

### 4.2.1 General

In the context of this specification, media to be rendered and displayed by the XR device through the XR runtime is typically available in an compressed form on the device. In contrast, media is accessed using a 5G System, decoded in the device using media capabilities, and the decoded media is rendered to then be provided through swap chains to the XR Runtime as shown in Figure 4.2.1-1.



Figure 4.2.1-1 Media pipelines: Access, decoding and rendering

The rendering function is responsible to adapt the content to be presentable by the XR Runtime by making use of a rendering loop and using swapchains. The application configures pipeline of different processes, namely the media access, the decoding and the rendering. The static information provided to the rendering step needs to be sufficient to configure the number of layers as well as each layer appropriately including

- View configuration

- Blend modes

- XR spaces

- swap chain formats and images

- projection layer types

[Frame rates: https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR\_FB\_display\_refresh\_rate

Rendering supported by the XR runtime

- Visual

- Audio]

### 4.2.2 Basic Media Pipeline

[Single media type

Access & Media decoder + Metadata + Render pose + Display time -> Swap Chain -> XR Runtime Composition (+ time warping), SEI Messages

Rendering is Conversion to RGB]

### 4.2.3 Advanced Media Pipelines

[Multiple decoders, VDI

Composition of multiple layers

Advanced Rendering (GPU Supported) – Scene Rendering (3D Rendering):

- Scene (Media decoder + Metadata) -> Vulkan API -> GPU + render pose è->Swap Chain -> XR Runtime Composition

Optional and mandatory formats – XR Runtime API supports capability query.]

### 4.2.4 Rendering capabilities

[To be defined]

## 4.3 Application and Service Provider view

[Usage of Capabilities in different delivery environments]

## 4.4 Structure of the specification

[Ed note: how to read this spec]

[ET: Probably arriving too late, before clause 4?]

# 5 Device reference architecture and interfaces

## 5.1 Architecture

The XR Baseline Client represents the functionalities, the peripherals, and the interfaces that are present on a generic XR UE. The actual device may be realized by a single device, or a combination of devices linked together. The details on how to instantiate an XR Baseline Client in the context of a service or deployment scenario is left for the respective Work Items and Study Items to define.



Figure 4.3.1-1 - XR Baseline terminal architecture

## 5.2 Description of the functional blocks

In terms of functionalities, an XR Baseline Client is composed of:

- An **XR application**: a software application that integrates audio-visual content into the user’s real-world environment

- An **XR Runtime**: a set of functions that interface with a platform to perform commonly required operations, such as accessing the controller/peripheral state, getting current and/or predicted tracking positions, performing spatial computing, as well as submitting rendered frames to the display processing unit and rendered audio to the speakers with a late stage re-projection to the latest pose.

- An **XR Source Management**: management of data sources provided through the XR runtime such as microphones, cameras, trackers, etc, for instance, making the information available to the XR application or providing it to the MAF for sending in the uplink.

- A **Media Access Function**: A set of functions that enables access to media and other XR-related data that is needed in the Scene manager or XR Runtime to provide an XR experience as well to create delivery formats for information provided by the XR Source Management.

- A **Scene Manager**: a set of functions that supports the application in arranging the logical and spatial representation of a multisensorial scene based on support from the XR Runtime.

- A **Presentation Engine**: a set of composite renderers, rendering the component of the scenes, based on the input from the Scene Manager.

- A **Media Session Handler**: a set of functions responsible for handling all 5G control plane operations, such as requesting network assistance, discovering and allocating edge resources, etc. This may be realized as a 5G-RTC MSH, 5GMS Media Session Handler, or any other functionIn addition, those functional blocks are integrated together via interfaces. Interfaces may be made of APIs and/or data formats and collectively act as a contract between the two sides of the interface.

In addition, those functional blocks are integrated together via interfaces. Interfaces may be made of APIs and/or data formats and collectively act as a contract between the two sides of the interface.

## 5.3 Interfaces and APIs

The XR Baseline Client contains the following interfaces:

- **IF-1** lies between the XR Runtime on one side and the Application (1a), the XR Source Management (1b) and the Presentation Engine (1c). IF-1 is implemented as an API (API-1) that exposes functions provided by the XR Runtime. An example of this API is the Khronos OpenXR API.

- **IF-2** describes the functions exposed by the XR Source Management that can be accessed and controlled by the XR application, or possibly other functions in the device. IF-2 is typically implemented as an API.

- **IF-3** lies between the XR Source Management and the Media Access Function and provides serialized information accessible on XR Runtime to the MAF.

- **IF-4** lies between the Media Access Function and the 5G System for user plane data.

- **IF-5** lies between the UE and the 5G System, implementing control sessions (such as 5G Media Streaming, IMS). This interface provides for instance the functionality of the RTC-5 interface as defined by TS 26.506 [REF].

- **IF-6** lies between the Media Session Handler and the Application/MAF. It offers the tools for them to activate 5G media functionality such as network assistance and edge resource discovery. The IF-6 is realized through an API (API-6).

- **IF-7** lies between the XR Application and the Media Access function to configure Media Access. This is typically implemented as an API (API-7) that exposes functions of the MAF.

- **IF-8** is an interface that allows the XR application to make use of 5G System connectivity.

- **IF-9** lies between the Scene Manager and the Media Access Function.

- **IF-10** lies between the Scene Manager and the XR Application.

# 6 General and systems functions and capabilities

[Ed note: Description of general functions such as sensors, runtime and their different capabilities, same for system aspects including protocols…]

## 6.1 Metadata formats

### 6.1.1 General

TBD

### 6.1.2 Pose Prediction Format

The split rendering client on the XR device periodically transmits a set of pose predictions to the split rendering server. The type of the message shall be set to “**urn:3gpp:split-rendering:v1:pose**”.

Each predicted pose shall contain the associated predicted display time and an identifier of the XR space that was used for that pose.

Depending on the view configuration of the XR session, there could be different pose information for each view.

The payload of the message shall be as follows:

Table 5.1.2-1 - Pose Prediction Format

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Cardinality | Description |
| poseInfo | Object | 1..n | An array of pose information objects, each corresponding to a target display time and XR space.  |
|  displayTime | number | 1..1 | The time for which the current view poses are predicted. |
|  xrSpace | number | 0..1 | An identifier for the XR space in which the view poses are expressed. The set of XR spaces are agreed on between the split rendering client and the split rendering server at the setup of the split rendering session. |
|  viewPoses | Object | 0..n | An array that provides a list of the poses associated with every view. The number of views is determined during the split rendering session setup between the split rendering client and server, depending on the view configuration of the XR session. |
|  pose | Object | 1..1 | An object that carries the pose information for a particular view. |
|  orientation | Object | 1..1 | Represents the orientation of the view pose as a quaternion based on the reference XR space. |
|  x | number | 1..1 | Provides the x coordinate of the quaternion. |
|  y | number | 1..1 | Provides the y coordinate of the quaternion. |
|  z | number | 1..1 | Provides the z coordinate of the quaternion. |
|  w | number | 1..1 | Provides the w coordinate of the quaternion. |
|  position | Object | 1..1 | Represents the location in 3D space of the pose based on the reference XR space. |
|  x | number | 1..1 | Provides the x coordinate of the position vector. |
|  y | number | 1..1 | Provides the y coordinate of the position vector. |
|  z | number | 1..1 | Provides the z coordinate of the position vector. |
|  fov | Object | 1..1 | Indicates the four sides of the field of view used for the projection of the corresponding XR view. |
|  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. |

### 6.1.3 Action Format

Actions are grouped into action sets, which may be activated and deactivated during the lifetime of an XR session. The action sets and actions are negotiated at the start of the split rendering session.

The split rendering client reports any changes to action state as soon as it occurs by sending a message of the type “**urn:3gpp:split-rendering:v1:action**”.

The content of the action message type shall follow the following format:

Table 5.1.3-1 - Action Format

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Cardinality | Description |
| actionSets | Object | 1..n | An array of active action sets, for which there is at least an action that has a state change.  |
|  actions | Object | 1..n | An array of objects that conveys information about the actions of the parent action set. |
|  identifier | string | 1..1 | A unique identifier of the action that was agreed upon during split rendering session setup. |
|  subactionPath | string | 1..1 | The sub-action path for which the state has changed. It abstracts a binding between an action and the hardware input associated to it by the XR runtime. |
|  state | object | 1..1 | The state of the action that had a change in state. |
|  lastChangeTime | number | 1..1 | The timestamp of the last change to the state of this action. |
|  currentStateBool | Bool | 0..1 | The current Boolean state of the action |
|  currentStateNum | number | 0..1 | The current numerical state of the action. |
|  currentStateVec2 | Array | 0..1 | An array of numerical state values for the action. |

# 7 Visual functions and capabilities

[Ed note: eg description of video formats and codecs, same for GPU capabilities and formats]

## 7.1 Decoding Capabilities

### 7.1.1 Video Decoding

**AVC-FullHD-Dec**: the capability to decode H.264 (AVC) Progressive High Profile Level 4.0 [6] bitstreams, with the chroma format being 4:2:0; and the bit depth being 8 bit.

**HEVC-FullHD-Dec**: the capability to decode H.265 (HEVC) Main10 Profile, Main Tier, Level 4.1[7] bitstreams that have general\_progressive\_source\_flag equal to 1, general interlaced\_source\_flag equal to 0, with the chroma format

### 7.1.2 Video decoding interface

**AVC-HEVC-2-Dec:** the capability to support two concurrent video decoding instances from any of the following profiles that are **AVC-FullHD-Dec** and **HEVC-FullHD-Dec**.

**AVC-HEVC-4-Dec:** the aggregate simultaneous processing of four video decoding instances of **HEVC-UHD-Dec** and **HEVC-8k-Dec.**

## 7.2 Encoding Capabilities

### 7.2.1 Video Encoding

**AVC-FullHD-Enc:** the capability to encode H.264 (AVC) Progressive High Profile Level 4.2 [6] bitstreams, with the chroma format being 4:2:0; and the bit depth being 8 bit.

**HEVC-FullHD-Enc:** the capability to encode a video signal to a bitstream that is decodable by a decoder that is **HEVC-FullHD-Dec** capable as defined in clause 6.1

## 7.3 Scene Processing Capabilities

## 7.4 Capability exchange

# 8 Audio functions and capabilities

[Ed note: eg description of audio formats and codecs and their associated capabilities]

# 9 QoE Metrics

[Editor’s note: related WID objectives

Identify which QoE metrics from VR QoE metrics can be reused or enhanced for AR media (e.g., resolution per eye, Field of view (FOV), round-trip interaction delay, etc.) and define relevant KPIs that are dedicated to AR/MR

Specify additional relevant KPIs and simple QoE Metrics for AR media]

# 10 Device Types and Media Profiles

## 10.1 Introduction

## 10.2 Device Type 1: Thin AR Glasses

Power-constrained

AR

The following XR Runtime functions are required to be supported

- Minimum pixel with and height of 1k by 1k per eye

- Frame rate: at least 60fps @1kx1k

- 16 Composition Layers

- Orientation and position tracking

- Blend mode: additive

- View configuration: mono, stereo

- Reference space: View, local, stage

- Swap chain formats: RGBA

- Swap Chain images: 2

- Rendering capabilities: YUV to RGB conversion

## 10.3 Device Type 2: AR Glasses

Power-constrained

AR

The following XR Runtime functions are required to be supported

- Minimum pixel with and height of 1k by 1k per eye

- Frame rate: at least 60fps @1kx1k

- 16 Composition Layers

- Orientation and position tracking

- Blend mode: additive, alpha\_blend

- View configuration: stereo

- Reference space: View, local, stage

- Swap chain formats: RGBA

- Swap Chain images: 2

- Rendering capabilities: YUV to RGB conversion

## 10.3 Device Type 3: XR Phone

Mono display

AR and VR

The following XR Runtime functions are required to be supported

- Minimum pixel with and height of 1k

- Frame rate: at least 60fps @1k

- 16 Composition Layers

- Orientation and position tracking

- Blend mode: alpha\_blend, additive, opaque

- View configuration: mono

- Reference space: View, local, stage

- Swap chain formats: RGBA

- Swap Chain images: 2

- Rendering capabilities: YUV to RGB conversion, advanced rendering

## 10.4 Device Type 4: XR HMD

Stereo display

AR and VR

The following XR Runtime functions are required to be supported

- Minimum pixel with and height of 2k by 2k per eye

- Frame rate: at least 60fps @2k

- 16 Composition Layers

- Orientation and position tracking

- Blend mode: alpha\_blend, additive, opaque

- View configuration: mono, stereo

- Reference space: View, local, stage

- Swap chain formats: RGBA

- Swap Chain images: 2

- Rendering capabilities: YUV to RGB conversion, advanced rendering

Annex A (informative/normative):
KPIs for AR/MR

# A.1 Introduction

[Editor’s note: related WID objectives

Identify which QoE metrics from VR QoE metrics can be reused or enhanced for AR media (e.g., resolution per eye, Field of view (FOV), round-trip interaction delay, etc.) and define relevant KPIs that are dedicated to AR/MR

Specify additional relevant KPIs and simple QoE Metrics for AR media]

Annex B (informative):
Usage of OpenXR [and WebXR] as XR Runtime

# B.1 Introduction

# B.2 Capability Mapping to OpenXR

### 4.1.4 XR Views and Rendering Loop

[ET: Can we move the following to the OpenXR annex?]

[To OpenXR annex START]

For visual rendering, the following applies:

1) To present images to the user, the runtime provides images organized in swapchains for the application to render into.

2) The XR Runtime may support different swapchain image formats and the supported image formats may be provided to the application through the runtime API. XR Runtimes typically support at least sRGB formats. Details may depend on the graphics API specified when creating the session.

3) *Swapchain* images may be 2D or 2D Array. Arrays allow to extract a subset of the 2D images for rendering. Multiple swapchain handles may exist simultaneously, up to some limit imposed by the XR runtime. Swap chain parameters include:

- texture format identifier, a graphics API specific version of a format, for example sRGB.

- width and height, expressing the pixel count of the images sent to the swapchain

- faceCount, being the number of faces, which can be either 6 (for cubemaps) or 1

- indication whether the swapchain is dynamic, i.e. updated as part of the XR rendering loop or static, i.e. the application releases only one image to this swapchain over its entire lifetime.

- access protection, indicating that the swapchain’s images are protected from CPU access

4) Once a session is running and in focussed state as introduced in clause 4.1.2, the following rendering loop is executed following Figure 4.1.4

a) The XR Application retrieves the action state, e.g. the status of the controllers and their associated pose. The application also establishes the location of different trackables.

b) Before an application can begin writing to a swapchain image, it first waits on the image to avoid writing to it before the Compositor has finished reading from it. Then an XR application synchronizes its rendering loop to the runtime. In the common case that an XR application has pipelined frame submissions, the application is expected to compute the appropriate target display time using both the predicted display time and predicted display interval. An XR Runtime is expected to provide and operate a swapchain that supports a specific frame rate.

c) Once the wait time completes, the application initiates the rendering process. In order to support the application in rendering different views the XR Runtime provides access to the viewer pose and projection parameters that are needed to render the different views. The view and projection info is provided for a particular display time within a specified XR space. Typically, the target/predicted display time for a given frame.

D) the application then performs its rendering work. Rendering work may be very simple, for example just directly copying data from the application into the swap chain or may be complex, for example iterating over the scene graph nodes and rendering complex objects. Once all views/layers are rendered, the application sends them to the XR Runtime for final compositing including the expected display time as well as the associated render pose.

e) An XR Runtime typically supports (i) planar projected images rendered from the eye point of each eye using a perspective projection, typically used to render the virtual world from the user’s perspective, and (ii) quad layer type describing a posable planar rectangle in the virtual world for displaying two-dimensional content. Other projection types such as cubemaps, equirectangular or cylindric projection may also be supported.

f) The XR application offloads the composition of the final image to an XR Runtime-supplied compositor. By this, the rendering complexity is significantly lower since details such as frame-rate interpolation and distortion correction are performed by the XR Runtime. It is assumed that the XR Runtime provides a compositor functionality for device mapping. A Compositor in the runtime is responsible for taking all the received layers, performing any necessary corrections such as pose correction and lens distortion, compositing them, and then sending the final frame to the display. An application may use multiple composition layers for its rendering. Composition layers are drawn in a specified order, with the 0th layer drawn first. Layers are drawn with a “painter’s algorithm,” with each successive layer potentially overwriting the destination layers whether or not the new layers are virtually closer to the viewer. Composition layers are subject to blending with other layers. Blending of layers can be controlled by layer per-texel source alpha. Layer swapchain textures may contain an alpha channel. Composition and blending is done in RGBA.

g) After the compositor has blended and flattened all layers, it then presents this image to the system’s display. The composited image is then blend with the user’s view of the physical world behind the displays in one of three modes, based on the application’s chosen environment blend mode:

- OPAQUE. The composition layers are displayed with no view of the physical world behind them. The composited image is interpreted as an RGB image, ignoring the composited alpha channel. This is the typical mode for VR experiences, although this mode can also be supported on devices that support video passthrough.

- ADDITIVE: The composition layers are additively blended with the real world behind the display. The composited image is interpreted as an RGB image, ignoring the composited alpha channel during the additive blending. This is the typical mode for an AR experience on a see-through headset with an additive display, although this mode can also be supported on devices that support video passthrough.

- ALPHA\_BLEND. The composition layers are alpha-blended with the real world behind the display. The composited image is interpreted as an RGBA image, with the composited alpha channel determining each pixel’s level of blending with the real world behind the display. This is the typical mode for an AR experience on a phone or headset that supports video passthrough.

h) Meanwhile, while the XR Runtime uses the submitted frame for compositing and display, a new rendering process may be kicked off for a different swap chain image.

[To OpenXR annex END]

# [B.3 Capability Mapping to WebXR]

Annex <X> (informative):
Change history

|  |
| --- |
| **Change history** |
| **Date** | **Meeting** | **TDoc** | **CR** | **Rev** | **Cat** | **Subject/Comment** | **New version** |
| 2022-04 | SA4#118e | S4-220504 |  |  |  | Draft TS sekeleton from the editor | 0.1.0 |
| 2023-04 | SA4#123-e |  |  |  |  |  | 0.2.0 |