**3GPP TSG SA WG4#121 S4-221578**

**Toulouse, 14th – 18th November 2022 revision of S4-221328**

|  |
| --- |
| *CR-Form-v12.0* |
| **PSEUDO CHANGE REQUEST** |
|  |
|  | **26**.**119-PD** | **CR** | **pseudo** | **rev** | **-** | **Current version:** | **3.1.0** |  |
|  |
| *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 | **X** | Radio Access Network |  | Core Network | **X** |

|  |
| --- |
|  |
| ***Title:***  | **[MeCAR] Minimum Device Architecture** |
|  |  |
| ***Source to WG:*** | Qualcomm Incorporated |
| ***Source to TSG:*** |  |
|  |  |
| ***Work item code:*** | **MeCAR** |  | ***Date:*** | 08/11/2022 |
|  |  |  |  |  |
| ***Category:*** | **B** |  | ***Release:*** | 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-8 (Release 8)Rel-9 (Release 9)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)* |
|  |  |
| ***Reason for change:*** | Until now the exact details on definition for the EDGAR framework were unclear. This document provides a refinement of the framework architecture and provides initial considerations on the assumptions of what to be supported by the runtime for visual rendering and composition, audio rendering and composition as well as XR runtime functions. |
|  |  |
| ***Summary of change:*** |  |
|  |  |
| ***Consequences if not approved:*** |  |
|  |  |
| ***Clauses affected:*** | 4.2.2 (new), 4.2.3 (new) |
|  |  |
|  | **Y** | **N** |  |  |
| ***Other specs*** |  | **X** |  Other core specifications  | TS/TR ... CR  |
| ***affected:*** |  | **X** |  Test specifications | TS/TR ... CR ...  |
| ***(show related CRs)*** |  | **X** |  O&M Specifications | TS/TR ... CR ...  |
|  |  |
| ***Other comments:*** |  |
|  |  |
| ***This CR's revision history:*** |  |

**===== CHANGE =====**

4.2.2 Thin AR UE

#### 4.2.2.1 Introduction

Based on the framework in clause 4.2.1, a simplified version of an AR device is presented following the principles for a 5G\_STAR EDGAR-type device architecture with a standalone 5G System integrated. It is referred to as a "Thin AR UE". In this case it is taken into account that the device is not capable to render complex 3D scenes and objects, but mainly makes use of the XR runtime.

In a typical use case, the media is pre-rendered for a specific time and render pose outside of the device, for example in the 3GPP network, and the Scene manager only converts the data to be compatible with the XR Runtime formats. Pre-renderig for video may be done to 2D video projections, possibly augmented with additional depth information (indicated as 2.5D in the figure). For audio, corresponding pre-rendering formats may be considered. In the uplink a coded representation of the 6DoF pose, sampled from XR Runtime, needs to be made available for prerendering to this pose. Such a UE may be used in a split rendering application.



Editor’s Note: the above diagram is expected to be further updated and refined based on the agreements of the XR baseline client.

**Figure 12 EDGAR UE device functions**

In the following, initial assumptions and potential requirements for the XR runtime for visual and audio processing are provided taking into account existing systems, in particular OpenXR, OpenGL ES and OpenSL ES. In all cases, the focus is on the functional methods of these specifications. Reference to specifics in these specifications does not imply that we mandate any of these specifications, but they serve as a reference.

Implementations may be done differently.

#### 4.2.2.2 XR Runtime and Source processing

For XR source processing, the following is assumed

* The application (including the scene manager) has access to the user pose and projection parameters that are needed to render the different scenes. The XR runtime provides the user pose and visual projection parameters needed for the visual rendering using a function equivalent to the OpenXR [xrLocateViews](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#xrLocateViews) function to render views for use in a composition projection layer. The xrLocateViews function returns the view and projection info for a particular playback/display time. This time is typically the target playback/display time for a given frame. Repeatedly calling xrLocateViews with the same time may not necessarily return the same result. 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 get the predicted views as late as possible in its pipeline to get the least amount of latency and prediction error. The user pose and visual projection parameters may need to be provided to the MAF.
* This specification does not define any requirements on input actions or haptics. However, input actions may be provided to the XR Source management to be delivered to the network. In summary
	+ A 6DoF predicted pose for a target playback/display time may be sampled from the XR Runtime at a frequency of at least 1kHz
	+ This information may be provided to a pose compressor that may send a compressed and quantized version to the network
* Audio or video sources may also be provided directly to the XR source manager.

#### 4.2.2.2 XR Visual Processing

For visual processing, OpenXR and OpenGL ES aligned terminology is used for reference, but this does not imply that we mandate any of these specifications. The following is assumed:

1. To present images to the user, the runtime provides images organized in swapchains for the application to render into. The XR runtime is expected to allow applications to create multiple swapchains (at least 4).
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 is expected to at least support R8G8B8A8 and R8G8B8A8 sRGB formats. Details may depend on the graphics API specified in xrCreateSession. Options include DirectX or OpenGL.
3. Support for OpenGL ES as a reference is assumed, i.e. an extension equivalent to the functionalities provided in [XR\_KHR\_opengl\_es\_enable](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_KHR_opengl_es_enable). OpenGL ES is platform independent and suited for embedded systems. The version and a subset of functionalities is still to be determined, likely 3.2. Again note that this is assumed as a reference.
4. Swapchain images can be 2D or 2D Array. Arrays allow to extract a subset of the 2D images for rendering.
5. The application or scene manager can offload the composition of the final image to a 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 these functionalities.
6. A runtime on a XR device is expected to support at least the equivalent functionalities of OpenXR composition, namely
	1. [XrCompositionLayerProjection](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XrCompositionLayerProjection): The projection layer type represents planar projected images rendered from the eye point of each eye using a perspective projection. This layer type is typically used to render the virtual world from the user’s perspective.
	2. [XrCompositionLayerQuad](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XrCompositionLayerQuad): The quad layer type describes a posable planar rectangle in the virtual world for displaying two-dimensional content. Quad layers can subtend a smaller portion of the display’s field of view, allowing a better match between the resolutions of the XrSwapchain image and footprint of that image in the final composition. This improves legibility for user interface elements or heads-up displays and allows optimal sampling during any composition distortion corrections the runtime might employ.
7. A runtime on an XR device may support additional OpenXR composition functionalities, namely
	1. [XR\_TYPE\_COMPOSITION\_LAYER\_CUBE\_KHR](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_KHR_composition_layer_cube): This extension adds an additional layer type that enables direct sampling from cubemaps. The cube layer is the natural layer type for hardware accelerated environment maps. Without updating the image source, the user can look all around, and the compositor can display what they are looking at without intervention from the application.
	2. [XR\_TYPE\_COMPOSITION\_LAYER\_CYLINDER\_KHR](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_KHR_composition_layer_cylinder): This extension adds an additional layer type where the XR runtime must map a texture stemming from a swapchain onto the inside of a cylinder section. It can be imagined much the same way a curved television display looks to a viewer. This is not a projection type of layer but rather an object-in-world type of layer, similar to XrCompositionLayerQuad. Only the interior of the cylinder surface must be visible; the exterior of the cylinder is not visible and must not be drawn by the runtime.
	3. [XR\_TYPE\_COMPOSITION\_LAYER\_EQUIRECT\_KHR](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_KHR_composition_layer_equirect) and [XR\_TYPE\_COMPOSITION\_LAYER\_EQUIRECT2\_KHR](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_KHR_composition_layer_equirect2): This extension adds an additional layer type where the XR runtime must map an equirectangular coded image stemming from a swapchain onto the inside of a sphere. The equirect layer type provides most of the same benefits as a cubemap, but from an equirect 2D image source. This image source is appealing mostly because equirect environment maps are very common, and the highest quality you can get from them is by sampling them directly in the compositor.
	4. [XR\_KHR\_composition\_layer\_depth](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_KHR_composition_layer_depth): This extension defines an extra layer type which allows applications to submit depth images along with color images in projection layers, i.e. XrCompositionLayerProjection. The XR runtime may use this information to perform more accurate reprojections taking depth into account. Use of this extension does not affect the order of layer composition as described in Compositing.
8. Each image that is provided to the runtime for rendering has assigned a *reference pose* defining the position and orientation of the projection in the reference frame of the associated space.
9. The runtime provides information about a predicted display time for the next time that the runtime predicts a composited frame will be displayed, i.e. using [xrFrameState](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XrFrameState) if in context to OpenXR
10. The composition may refer to a sub-image as for example defined in [XrSwapchainSubImage](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XrSwapchainSubImage), i.e. representing the valid portion of the image to use, in pixels. It also implicitly defines the transform from normalized image coordinates into pixel coordinates.

#### 4.2.2.3 XR Audio Processing

Editor’s Note: The following section is only tentative and requires futher discussion with audio experts

For audio processing, OpenXR and OpenSL ES aligned terminology is used for reference, but this does not imply that we mandate any of these specifications. The following is aimed to reflect a typical decomposition of steps for immersive audio split rendering:

* An interface to the XR runtime is available hand over raw audio buffers to determine how the XR application and scene manager would access a device’s audio capabilities. In order address a concrete implementation example, the model of [OpenSL ES](https://www.khronos.org/opensles/) is used for reference. OpenSL ES supports both file-based and in-memory data sources, as well as buffer queues, for efficient streaming of audio data from memory to the audio system. Buffer queues in OpenSL may be seen corresponding to visual swap chains. OpenSL ES may be viewed as a companion to 3D graphic APIs such as OpenGL ES. The 3D graphics engine render the 3D graphics scene to a two-dimension display device, and the OpenSL ES implementation render the 3D audio scene to the audio output device.
* 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
	1. Non-diegetic, i.e. they are not rendered according to the pose.
	2. Diegetic, i.e. rendered according to the pose, and describing a full 6DoF experience in the reference space of the XR session. In this case, the XR runtime will create a rendered signal according to the latest pose.
	3. Diegetic and pre-rendered (outside the thin UE) for a specific render pose. In this case, the signals have been prepared such that the XR runtime can use the audio signal with their associated render pose and supplementary data for a pose correction to the latest pose.
	4. a mixture of such signals that are jointly presented.
	5. the signals may originate from different sources, for example some may be generated locally, others may be part of a pre-rendering or a full scene created in the network.
* Of particular interest is a simple use case, for example prominent for split rendering is a combination of signals a) non-diegetic, c) pre-rendered with assigned render pose, d) mixture of those.
* It is generally desirable to define reference renderers that can handle the rendering of a mix of audio signals as listed under a) to e).
* In terms of requirements for XR in general and MeCAR in particular, audio formats that address different use cases are needed.
* At least a), b), d) and e) from above follows the principle of what is defined in TS 26.118, figure 4.5-1 providing a Block diagram of Common Informative Binaural Renderer. Pre-processing of content to HOA may for example be done based on TS 26.118, Annex B.3. The displacement of the scene to adjust to the latest head pose may be based on what is described in TS 26.118, Annex B.4. The headphone output signal computation may follow TS 26.118, Annex B.5. As a starting point, the renderer in Annex B of TS 26.118 could be considered as an example renderer. However, note that this renderer may have certain limitations and, for example, not be fully usable for translational movements as well as to address signal types c) from above. Detailed analysis from audio experts is needed to assess if Annex B renderer is relevant for reference.

A renderer following these principles may be applicable for split rendering use cases which is a combination of signals a) non-diegetic, c) diegetic, pre-rendered with assigned render pose, d) mixture of those. Audio formats being under consideration includes, but is not limited to:

* + Mono signal (non-diegetic signals) that is not adapted to the latest viewer pose and position
	+ Stereo signal (non-diegetic signals) that is not adapted to the latest viewer pose and position
	+ Stereo signal (diegetic signal) with pre-rendered pose (e.g. binaurally rendered)
	+ HOA signal with pre-rendered pose (for example the position) which may be rotated to the latest pose (pre-dominantly orientation).

****

Audio experts are invited to study and define audio formats and example renderers that can address the above use cases, taking into consideration interfaces as provided in OpenSL ES.

Addressing full 6DoF audio scenes with rendering on the device may however be subject of future work.

**===== CHANGE =====**

4.2.3 Advanced AR UE

A render centric-UE would support additional rendering capabilities in the device such as more complex 3D rendering (meshes and point clouds), uplink media capturing and so on. The basic principle of what is described in clause 4.2.2 remain. The additional capabilities may be expressed initially as optional extensions to the minimum capabilities in clause 4.2.2. No additional device classes are needed for now.

However, such extensions then would need to be done by identifying and adding the relevant capability checks.



**Figure 13 – Advanced AR UE**