**Source:** China Unicom, Qualcomm Incorporated

**Title:** Discussion on the Observation Points Monitoring

**Document for** Discussion and Agreement

**Agenda item:** 9.8- FS\_ARMRQoE (Feasibility Study on AR and MR QoE Metrics)

# Introduction

In the last SA4#122 meetings, the AR/MR QoE study item has defined Observation Points based on the AR architecture in MeCar WI, but the detailed metrics that can be observed at each Observation Point are TBD. This paper presents some observable parameters based on the runtime component, with reference to the OpenXR standard [1].

This contribution proposes to define the detailed parameters that can be observed by the observation point 1 defined in TR 26.812 v0.3 [2].

# Observable parameters for OP1

In the TR 26.812, Observation point 1 is derived from the XR Runtime API which exchanges information between XR Runtime and XR Source Management/Presentation Engine and is defined to monitor. In addition, MeCar PD v5.1 [3] also clarify that the OP1 (can also called IF1) is implemented as an API-1 that exposes functions provided by the XR Runtime. An example of this API is the Khronos OpenXR API. So the key is to prove the following parameters can be exposed (or output) through the runtime.

[Editor’s Note: the applicability of these parameters is TBD

- Viewer pose

- Projection parameters

- Camera information

- Gesture

- Body action

- Tracking position prediction error

- Mapping latency for reconstructing the surrounding space]

## Viewer pose and Projection parameters

Viewer pose is to present the user position and orientation, which can be defined as quaternion (X, Y, Z, W) for orientation and three vectors (X, Y, Z in cartesian coordinate system) for position [3][4]. Projection parameters are parameters associated to the perspective/orthogonal/omnidirectional projection to the 3D scene [3].

It’s noted that OpenXR is the interface between an application and an in-process or out-of-process "XR runtime system", or just "runtime" hereafter [1]. In OpenXR [1], an XR application uses *xrLocateViews* to retrieve the viewer pose and projection parameters needed to render each view for use in a composition projection layer. *xrLocateViews* returns an array of *XrView* elements and the *XrView* data structure is defined below:



Figure 2.1-1: XrView structure [1]

In *XrView* structure, it’s defined that pose is an *XrPosef* indicating the location and orientation of the view in the space specified by the *xrLocateViews* function, *fov* is the *XrFovf* for the four sides of the projection. And it also clarifies the *XrView* structure contains view pose and projection state necessary to render a single projection view in the view configuration.

Viewer pose and projection parameters can be monitored or observed via the OP1.

## Camera information

Camera information including the attribute of the camera and everything external to the camera, such as resolution, FOV, relative pose, attached to,etc [1].

The section 12.117 of XR\_OCULUS\_external\_camera in OpenXR [1] clarifies this extension enables the querying of external camera information for a session. This extension is intended to enable mixed reality capture support for applications. For details,

***XR\_OCULUS\_external\_camera*** API supports returning camera intrinsics and extrinsics.

The intrinsic parameters are the attributes of the camera and include:

* fov is the [XrFovf](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22XrFovf) for this camera’s viewport.
* virtualNearPlaneDistance is the near plane distance of the virtual camera used to match the external camera
* virtualFarPlaneDistance is the far plane distance of the virtual camera used to match the external camera
* imageSensorPixelResolution is the [XrExtent2Di](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22XrExtent2Di) specifying the camera’s resolution (in pixels).

The extrinsic parameters are everything external to the camera: relative pose, attached to, etc.

Camera information parameter, including the camera intrinsic and extrinsic, can be monitored or observed via the OP1.

## Gesture

Gesture can trigger specific actions during an AR experience, it can be provided as a list of hand joint poses which represent the current configuration of the tracked hands.

Clause 12.30, [XR\_EXT\_hand\_tracking](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html%22%20%5Cl%20%22XR_EXT_hand_tracking) in OpenXR [1] enables applications to locate the individual joints of hand tracking inputs. It enables applications to render hands in XR experiences and interact with virtual objects using hand joints.

The section 12.57, XR\_FB\_hand\_tracking\_aim in OpenXR [1], clarifies that the *XR\_EXT\_hand\_tracking* extension provides a list of hand joint poses which represent the current configuration of the tracked hands. This extension adds a layer of gesture recognition that is used by the system. That means an application is allowed to get a set of basic gesture states for the hand when using the *XR\_EXT\_hand\_tracking* extension.Hand gesture parameter can be monitored or observed via the OP1.

## Body action

Body action parameters includes body joints and joint locations. The section 12.44 of XR\_FB\_body\_tracking in OpenXR [1] clarifies that this extension enables applications to locate the individual body joints that represent the estimated position of the user of the device. It enables applications to render the upper body in XR experiences. When create a body tracker handle, this handle can be used to locate body joints using *xrLocateBodyJointsFB* function, and a body tracker provides joint locations with an unobstructed range of human body motion.

Body action parameters can be monitored or observed via the OP1.

## Tracking position prediction error

Tracking position prediction error represents the difference or gap between the predicted spaces locations and the target locations.

Section 7.4 of Locating Spaces in OpenXR [1] clarifies that applications use the *xrLocateSpace* function to find the pose of an XrSpace’s origin within a base XrSpace at a given historical or predicted time.

The structure of *xrLocateSpace* is describe as blow:



Figure 2.5-1: *xrLocateSpace* structure [1]

The detailed parameters description are listed in figure 2.5-2:



Figure 2.5-2: Parameters Description [1]

It also described that for a time in the past, the runtime should locate the spaces based on the runtime’s most accurate current understanding of how the world was at that historical time. For a time in the future, the runtime should locate the spaces based on the runtime’s most up-to-date prediction of how the world will be at that future time. The minimum valid range of values for time are described in Prediction Time Limits. With respect to backward prediction, the application can pass a prediction time equivalent to the timestamp of the most recently received pose plus as much as 50 milliseconds in the past to retrieve accurate historical data.

To summarize clause 2.5, if the predicated spaces locations (or positions) based on the historical time or future time are not the same as (or accurate) as the target locations that the runtime want to locate, the error will happen and it can be monitored by comparing the predicated spaces locations and target locations. So it can prove that the tracking position prediction error parameter can be monitored or observed via the OP1.

# Proposal

We propose to agree that Viewer pose, Projection parameters, Camera information, Gesture, Body action, and Tracking position prediction error parameters can be monitored by the observation point 1 and capture it into the TR 26.812.

# References

1. The OpenXR Specification, Copyright (c) 2017-2023, The Khronos Group Inc., Version 1.0.27: from git ref release-1.0.27
2. S4-230294 TR 26812\_030
3. S4aV230017 - MeCAR Permanent Document v5.1
4. S4-230393 5G\_RTP Permanent Document v. 0.0.4